A Markdown Interpreter for TEX

Vít Novotný
witiko@mail.muni.cz

Version 2.17.0-0-g6428569
2022-09-30

Contents

List of Figures

1 Introduction

The Markdown package[1] converts markdown[2] markup to TEX commands. The functionality is provided both as a Lua module and as plain TEX, LATEX, and ConTEXt macro packages that can be used to directly typeset TEX documents containing markdown markup. Unlike other convertors, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😉

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the

[1] See https://ctan.org/pkg/markdown.
[2] See https://daringfireball.net/projects/markdown/basics.

implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.[3]

```
local metadata = {
    version   = "(((VERSION)))",
    comment   = "A module for the conversion from markdown to plain TeX",
    author    = "John MacFarlane, Hans Hagen, Vít Novotný",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2022 Vít Novotný"},
    license   = "LPPL 1.3c"
}

if not modules then modules = { } end
modules['markdown'] = metadata
```

1.1 Requirements

This section gives an overview of all resources required by the package.

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTEX engine:

LPeg $\geqslant$ 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg $\geqslant$ 0.10 is included in LuaTEX $\geqslant$ 0.72.0 (TEXLive $\geqslant$ 2013).

```
local lpeg = require("lpeg")
```

Selene Unicode A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and footnote tags to the lower case. Selene Unicode is included in all releases of LuaTEX (TEXLive $\geqslant$ 2008).

```
local unicode
(function()
  local ran_ok
  ran_ok, unicode = pcall(require, "unicode")
```

If the Selene Unicode library is unavailable and we are using Lua $\geqslant$ 5.3, we will use the built-in support for Unicode.

[3] See http://mirrors.ctan.org/macros/generic/markdown/markdown.html.

```
  if not ran_ok then
    unicode = {["utf8"]={char=utf8.char}}
  end
end)()
```

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaTeX (TeXLive ⩾ 2008).

```
local md5 = require("md5")
```

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [1, Section 4.3]. Beside these, we also carry the following third-party Lua libraries:

api7/lua-tinyyaml A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jekyllData` option is enabled.

1.1.2 Plain TeX Requirements

The plain TeX part of the package requires that the plain TeX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.1.1), and the following packages:

expl3 A package that enables the expl3 language from the LaTeX3 kernel in TeX Live ⩽ 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
⟨@@=markdown⟩
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic\relax
\fi
```

lt3luabridge A package that allows us to execute Lua code with LuaTeX as well as with other TeX engines that provide the *shell escape* capability, which allows them to execute code with the system's shell.

The plain TeX part of the package also requires the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the `\markdownOptionCacheDir` macro before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive ⩾ 2008).

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [1, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [1, Section 4.3].

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.6), the plain TeX part of the package will require that either the LuaTeX `\directlua` primitive or the shell access file stream 18 is available in your TeX engine. If only the shell access file stream is available in your TeX engine (as is the case with pdfTeX and XeTeX) or if you enforce the use of shell using the `\markdownMode` macro, then unless your TeX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 LaTeX Requirements

The LaTeX part of the package requires that the LaTeX 2$_\varepsilon$ format is loaded,

```
26 \NeedsTeXFormat{LaTeX2e}%
```

a TeX engine that extends ε-TeX, and all the plain TeX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.4 and 3.3.4) or LaTeX themes (see Section 2.3.2.2) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.1):

url A package that provides the `\url` macro for the typesetting of links.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists.

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` LaTeX theme (see Section 2.3.2.2), and to provide default token renderer prototypes.

fancyvrb A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code.

csvsimple A package that provides the `\csvautotabular` macro for typesetting CSV files in the default renderer prototypes for iA,Writer content blocks.

gobble A package that provides the `\@gobblethree` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive $\geqslant$ 2016.

amsmath and amssymb Packages that provide symbols used for drawing ticked and unticked boxes.

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` LaTeX theme, see Section 2.3.2.2.

grffile A package that extends the name processing of package graphics to support a larger range of file names in 2006 ⩽ TeX Live ⩽ 2019. Since TeX Live ⩾ 2020, the functionality of the package has been integrated in the LaTeX 2_ε kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` LaTeX themes, see Section 2.3.2.2.

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.6, and also in the default renderer prototype for attribute identifiers.

soulutf8 A package that is used in the default renderer prototype for strike-throughs.

```
\RequirePackage{expl3}
```

1.1.4 ConTeXt Prerequisites

The ConTeXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain TeX prerequisites (see Section 1.1.2), and the following ConTeXt modules:

m-database A module that provides the default token renderer prototype for iA,Writer content blocks with the CSV filename extension (see Section 2.2.4).

1.2 Feedback

Please use the Markdown project page on GitHub[4] to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the TeX-LaTeX Stack Exchange.[5] community question answering web site under the `markdown` tag.

1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

[4] See `https://github.com/witiko/markdown/issues`.
[5] See `https://tex.stackexchange.com`.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

Support for content slicing (Lua options `shiftHeadings` and `slice`) and pipe tables (Lua options `pipeTables` and `tableCaptions`) was graciously sponsored by David Vins and Omedym.

The TEX implementation of the package draws inspiration from several sources including the source code of LATEX 2_ε, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from TEX, the filecontents package by Scott Pakin and others.

2 Interfaces

This part of the documentation describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package.

Since neither TEX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is exposed by the Lua layer. The plain TEX layer exposes the conversion capabilities of Lua as TEX macros. The LATEX and ConTEXt layers provide syntactic sugar on top of plain TEX macros. The user can interface with any and all layers.

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain TEX. This interface is used by the plain TEX implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

The Lua interface is implemented by the `markdown` Lua module.

```
28 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain TEX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain TEX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

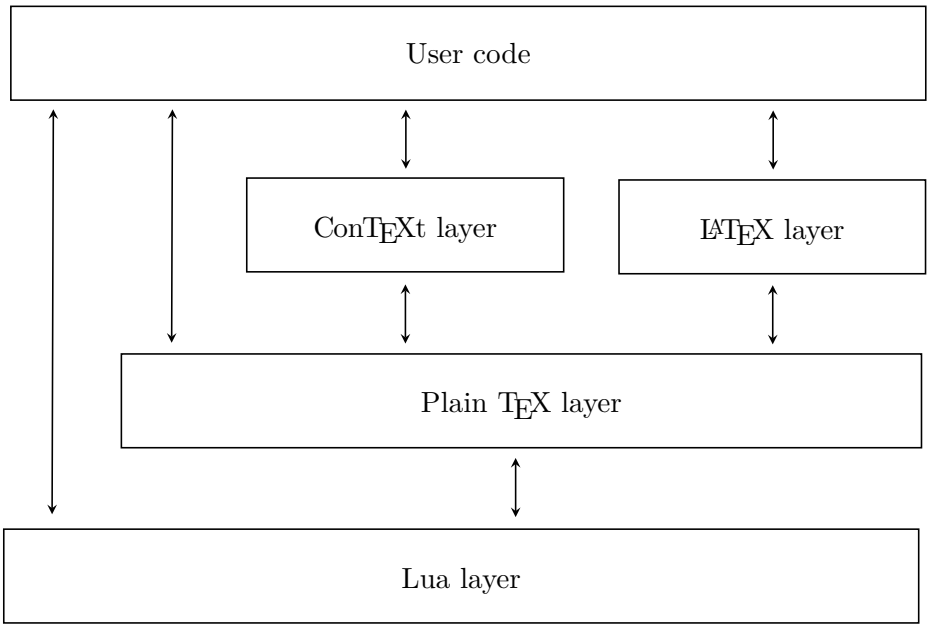

Figure 1: A block diagram of the Markdown package

The following example Lua code converts the markdown string `Hello *world*!` to a TEX output using the default options and prints the TEX output:

```
local md = require("markdown")
local convert = md.new()
print(convert("Hello *world*!"))
```

2.1.2 User-Defined Syntax Extensions

For the purpose of user-defined syntax extensions, the Lua interface also exposes the `reader` object, which performs the lexical and syntactic analysis of markdown text and which exposes the `reader->insert_pattern` and `reader->add_special_character` methods for extending the PEG grammar of markdown.

The read-only `walkable_syntax` hash table stores those rules of the PEG grammar of markdown that can be represented as an ordered choice of terminal symbols. These rules can be modified by user-defined syntax extensions.

```
29 local walkable_syntax = {
30   Block = {
31     "Blockquote",
32     "Verbatim",
```

```
    "HorizontalRule",
    "BulletList",
    "OrderedList",
    "Heading",
    "DisplayHtml",
    "Paragraph",
    "Plain",
  },
  Inline = {
    "Str",
    "Space",
    "Endline",
    "UlOrStarLine",
    "Strong",
    "Emph",
    "Link",
    "Image",
    "Code",
    "AutoLinkUrl",
    "AutoLinkEmail",
    "AutoLinkRelativeReference",
    "InlineHtml",
    "HtmlEntity",
    "EscapedChar",
    "Smart",
    "Symbol",
  },
}
```

The `reader->insert_pattern` method inserts a PEG pattern into the grammar of markdown. The method receives two arguments: a selector string in the form "⟨*left-hand side terminal symbol*⟩ ⟨*before*, *after*, *or instead of*⟩ ⟨*right-hand side terminal symbol*⟩" and a PEG pattarn to insert. For example. if we'd like to insert `pattern` into the grammar between the `Inline -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern` with `"Inline after Emph"` (or `"Inline before Link"`) and `pattern` as the arguments.

The `reader->add_special_character` method adds a new character with special meaning to the grammar of markdown. The method receives the character as its only argument.

2.1.3 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` table.

```
local defaultOptions = {}
```

To enable the enumeration of Lua options, we will maintain the \g_@@_lua_options_seq sequence.

```
\ExplSyntaxOn
\seq_new:N \g_@@_lua_options_seq
```

To enable the reflection of default Lua options and their types, we will maintain the \g_@@_default_lua_options_prop and \g_@@_lua_option_types_prop property lists, respectively.

```
\prop_new:N \g_@@_lua_option_types_prop
\prop_new:N \g_@@_default_lua_options_prop
\seq_new:N \g_@@_option_layers_seq
\tl_const:Nn \c_@@_option_layer_lua_tl { lua }
\seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
\cs_new:Nn
  \@@_add_lua_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_lua_tl
      { #1 }
      { #2 }
      { #3 }
  }
\cs_new:Nn
  \@@_add_option:nnnn
  {
    \seq_put_right:cn
      { g_@@_ #1 _options_seq }
      { #2 }
    \prop_put:cnn
      { g_@@_ #1 _option_types_prop }
      { #2 }
      { #3 }
    \prop_put:cnn
      { g_@@_default_ #1 _options_prop }
      { #2 }
      { #4 }
    \@@_typecheck_option:n
      { #2 }
  }
\cs_generate_variant:Nn
  \@@_add_option:nnnn
  { Vnnn }
\tl_const:Nn \c_@@_option_value_true_tl  { true  }
\tl_const:Nn \c_@@_option_value_false_tl { false }
\cs_new:Nn \@@_typecheck_option:n
  {
    \@@_get_option_type:nN
```

```
      { #1 }
      \l_tmpa_tl
    \str_case_e:Vn
      \l_tmpa_tl
      {
        { \c_@@_option_type_boolean_tl }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \bool_if:nF
              {
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_true_tl ||
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_false_tl
              }
              {
                \msg_error:nnnV
                  { @@ }
                  { failed-typecheck-for-boolean-option }
                  { #1 }
                  \l_tmpa_tl
              }
          }
      }
  }
\msg_new:nnn
  { @@ }
  { failed-typecheck-for-boolean-option }
  {
    Option~#1~has~value~#2,~
    but~a~boolean~(true~or~false)~was~expected.
  }
\cs_generate_variant:Nn
  \str_case_e:nn
  { Vn }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnnV }
\seq_new:N \g_@@_option_types_seq
\tl_const:Nn \c_@@_option_type_clist_tl { clist }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
\tl_const:Nn \c_@@_option_type_counter_tl { counter }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
```

```
\tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
\tl_const:Nn \c_@@_option_type_number_tl  { number  }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
\tl_const:Nn \c_@@_option_type_path_tl    { path    }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
\tl_const:Nn \c_@@_option_type_slice_tl   { slice   }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
\tl_const:Nn \c_@@_option_type_string_tl  { string  }
\seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
\cs_new:Nn
  \@@_get_option_type:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_ ##1 _option_types_prop }
          { #1 }
          \l_tmpa_tl
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { @@ }
          { undefined-option }
          { #1 }
      }
    \seq_if_in:NVF
      \g_@@_option_types_seq
      \l_tmpa_tl
      {
        \msg_error:nnnV
          { @@ }
          { unknown-option-type }
          { #1 }
          \l_tmpa_tl
      }
    \tl_set_eq:NN
```

```
      #2
      \l_tmpa_tl
  }
\msg_new:nnn
  { @@ }
  { unknown-option-type }
  {
    Option~#1~has~unknown~type~#2.
  }
\msg_new:nnn
  { @@ }
  { undefined-option }
  {
    Option~#1~is~undefined.
  }
\cs_new:Nn
  \@@_get_default_option_value:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_default_ ##1 _options_prop }
          { #1 }
          #2
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { @@ }
          { undefined-option }
          { #1 }
      }
  }
\cs_new:Nn
  \@@_get_option_value:nN
  {
    \@@_option_tl_to_csname:nN
      { #1 }
```

```
      \l_tmpa_tl
    \cs_if_free:cTF
      { \l_tmpa_tl }
      {
        \@@_get_default_option_value:nN
          { #1 }
          #2
      }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpa_tl
        \str_if_eq:NNTF
          \c_@@_option_type_counter_tl
          \l_tmpa_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nx
              #2
              { \the \cs:w \l_tmpa_tl \cs_end: }
          }
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nv
              #2
              { \l_tmpa_tl }
          }
      }
  }
\cs_new:Nn \@@_option_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownOption
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
```

2.1.4 File and Directory Names

`cacheDir`=⟨*path*⟩ default: `.`

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain TEX implementations. The Lua implementation expects that the entire path already exists.

When iteratively writing and typesetting a markdown document, the cache files are going to accumulate over time. You are advised to clean the cache directory every now and then, or to set it to a temporary filesystem (such as `/tmp` on UN*X systems), which gets periodically emptied.

```
\@@_add_lua_option:nnn
  { cacheDir }
  { path }
  { \markdownOptionOutputDir / _markdown_\jobname }
defaultOptions.cacheDir = "."
```

`frozenCacheFileName`=⟨*path*⟩ default: `frozenCache.tex`

A path to an output file (frozen cache) that will be created when the `finalizeCache` option is enabled and will contain a mapping between an enumeration of markdown documents and their auxiliary cache files.

The frozen cache makes it possible to later typeset a plain TEX document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain TEX option. As a result, the plain TEX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
\@@_add_lua_option:nnn
  { frozenCacheFileName }
  { path }
  { \markdownOptionCacheDir / frozenCache.tex }
defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.5 Parser Options

`blankBeforeBlockquote`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following blockquote.

`false` Do not require a blank line between a paragraph and the following blockquote.

```
\@@_add_lua_option:nnn
  { blankBeforeBlockquote }
  { boolean }
  { false }
defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following fenced code block.

`false` Do not require a blank line between a paragraph and the following fenced code block.

```
\@@_add_lua_option:nnn
  { blankBeforeCodeFence }
  { boolean }
  { false }
defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeHeading`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following header.

`false` Do not require a blank line between a paragraph and the following header.

```
\@@_add_lua_option:nnn
  { blankBeforeHeading }
  { boolean }
  { false }
defaultOptions.blankBeforeHeading = false
```

`breakableBlockquotes`=true, false default: `false`

`true` A blank line separates block quotes.

`false` Blank lines in the middle of a block quote are ignored.

```
\@@_add_lua_option:nnn
  { breakableBlockquotes }
  { boolean }
  { false }
defaultOptions.breakableBlockquotes = false
```

`citationNbsps`=true, false default: `false`

`true` Replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

`false` Do not replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

```
\@@_add_lua_option:nnn
  { citationNbsps }
  { boolean }
  { true }
defaultOptions.citationNbsps = true
```

`citations`=true, false default: `false`

`true` Enable the Pandoc citation syntax extension:

```
Here is a simple parenthetical citation [@doe99] and here
is a string of several [see @doe99, pp. 33-35; also
@smith04, chap. 1].

A parenthetical citation can have a [prenote @doe99] and
a [@smith04 postnote]. The name of the author can be
suppressed by inserting a dash before the name of an
author as follows [-@smith04].

Here is a simple text citation @doe99 and here is
a string of several @doe99 [pp. 33-35; also @smith04,
chap. 1]. Here is one with the name of the author
suppressed -@doe99.
```

`false` Disable the Pandoc citation syntax extension.

```
\@@_add_lua_option:nnn
  { citations }
  { boolean }
  { false }
defaultOptions.citations = false
```

`codeSpans`=`true`, `false` default: `true`

`true` Enable the code span syntax:

```
Use the `printf()` function.
``There is a literal backtick (`) here.``
```

`false` Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

```
``This is a quote.''
```

```
\@@_add_lua_option:nnn
  { codeSpans }
  { boolean }
  { true }
defaultOptions.codeSpans = true
```

`contentBlocks`=`true`, `false` default: `false`

`true` Enable the iA,Writer content blocks syntax extension [3]:

```
http://example.com/minard.jpg (Napoleon's
  disastrous Russian campaign of 1812)
/Flowchart.png "Engineering Flowchart"
/Savings Account.csv 'Recent Transactions'
/Example.swift
/Lorem Ipsum.txt
```

`false` Disable the iA,Writer content blocks syntax extension.

```
\@@_add_lua_option:nnn
  { contentBlocks }
  { boolean }
  { false }
defaultOptions.contentBlocks = false
```

`contentBlocksLanguageMap`=⟨*filename*⟩
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA,Writer content blocks. See Section 2.2.3.11 for more information.

```
\@@_add_lua_option:nnn
  { contentBlocksLanguageMap }
  { path }
  { markdown-languages.json }
defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`definitionLists`=`true`, `false` default: `false`

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

```
\@@_add_lua_option:nnn
  { definitionLists }
  { boolean }
  { false }
defaultOptions.definitionLists = false
```

`eagerCache`=`true`, `false` default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

This behavior will always be used if the `finalizeCache` option is enabled.

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

This behavior will only be used when the `finalizeCache` option is disabled. Recursive nesting of markdown document fragments is undefined behavior when `eagerCache` is disabled.

```
\@@_add_lua_option:nnn
  { eagerCache }
  { boolean }
  { true }
defaultOptions.eagerCache = true
```

`extensions`=⟨*filenames*⟩

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the kpathsea library is available, files will be searched for not only in the current working directory but also in the TEX directory structure.

A user-defined syntax extension is a Lua file in the following format:

```
local strike_through = {
  api_version = 1,
  grammar_version = 1,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
    local function between(p, starter, ender)
      ender = lpeg.B(nonspacechar) * ender
      return (starter * #nonspacechar
              * lpeg.Ct(p * (p - ender)^0) * ender)
    end

    local read_strike_through = between(
      lpeg.V("Inline"), doubletildes, doubletildes
    ) / function(s) return {"\\st{", s, "}"} end

    reader.insert_pattern("Inline after Emph", read_strike_through)
    reader.add_special_character("/")
  end
}

return strike_through
```

The `api_version` and `grammar_version` fields specify the version of the user-defined syntax extension API and the markdown grammar for which the extension was written. See the current API and grammar versions below:

```
metadata.user_extension_api_version = 1
metadata.grammar_version = 1
```

```
Any changes to the syntax extension API or grammar will cause the
corresponding current version to be incremented.  After Markdown 3.0.0,
any changes to the API and the grammar will be either backwards-compatible
or constitute a breaking change that will cause the major version of the
Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of
markdown using the interface of a Lua \luamref{reader} object, such as
the \luamref{reader->insert_pattern} and
\luamref{reader->add_special_character} methods,
see Section <#luauserextensions>.
```

```
\cs_generate_variant:Nn
  \@@_add_lua_option:nnn
  { nnV }
\@@_add_lua_option:nnV
  { extensions }
  { clist }
  \c_empty_clist
defaultOptions.extensions = {}
```

`expectJekyllData`=`true`, `false` default: `false`

`false` When the `jekyllData` option is enabled, then a markdown document may begin with YAML metadata if and only if the metadata begin with the end-of-directives marker (`---`) and they end with either the end-of-directives or the end-of-document marker (`...`):

```
\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
```

```
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}
```

`true` When the `jekyllData` option is enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

```
\documentclass{article}
\usepackage[jekyllData, expectJekyllData]{markdown}
\begin{document}
\begin{markdown}
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}
```

```
\@@_add_lua_option:nnn
  { expectJekyllData }
  { boolean }
  { false }
defaultOptions.expectJekyllData = false
```

`fancyLists`=`true`, `false` default: `false`

`true` Enable the Pandoc fancy list extension:

```
a) first item
b) second item
c) third item
```

`false` Disable the Pandoc fancy list extension.

```
\@@_add_lua_option:nnn
  { fancyLists }
  { boolean }
  { false }
defaultOptions.fancyLists = false
```

`fencedCode`=`true`, `false` default: `false`

`true` Enable the commonmark fenced code block extension:

````
~~~ js
if (a > 3) {
    moveShip(5 * gravity, DOWN);
}
~~~~~~

  ``` html
  <pre>
    <code>
      // Some comments
      line 1 of code
      line 2 of code
      line 3 of code
    </code>
  </pre>
  ```
````

`false` Disable the commonmark fenced code block extension.

```
\@@_add_lua_option:nnn
  { fencedCode }
  { boolean }
  { false }
defaultOptions.fencedCode = false
```

`finalizeCache`=`true`, `false` default: `false`

Whether an output file specified with the `frozenCacheFileName` option (frozen cache) that contains a mapping between an enumeration of markdown documents and their auxiliary cache files will be created.

The frozen cache makes it possible to later typeset a plain TEX document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` plain TEX option. As a result, the plain TEX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
\@@_add_lua_option:nnn
  { finalizeCache }
  { boolean }
  { false }
defaultOptions.finalizeCache = false
```

`footnotes`=`true`, `false` default: `false`

`true` Enable the Pandoc footnote syntax extension:

```
Here is a footnote reference,[^1] and another.[^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they
belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the
    first line.  In this way, multi-paragraph footnotes
    work like multi-paragraph list items.

This paragraph won't be part of the note, because it
isn't indented.
```

`false` Disable the Pandoc footnote syntax extension.

```
\@@_add_lua_option:nnn
  { footnotes }
```

```
  { boolean }
  { false }
defaultOptions.footnotes = false
```

`frozenCacheCounter`=⟨*number*⟩ default: `0`

The number of the current markdown document that will be stored in an output file (frozen cache) when the `finalizeCache` is enabled. When the document number is 0, then a new frozen cache will be created. Otherwise, the frozen cache will be appended.

Each frozen cache entry will define a TEX macro `\markdownFrozenCache`⟨*number*⟩ that will typeset markdown document number ⟨*number*⟩.

```
\@@_add_lua_option:nnn
  { frozenCacheCounter }
  { counter }
  { 0 }
defaultOptions.frozenCacheCounter = 0
```

`hardLineBreaks`=`true`, `false` default: `false`

`true` Interpret all newlines within a paragraph as hard line breaks instead of spaces.

`false` Interpret all newlines within a paragraph as spaces.

```
\@@_add_lua_option:nnn
  { hardLineBreaks }
  { boolean }
  { false }
defaultOptions.hardLineBreaks = false
```

`hashEnumerators`=`true`, `false` default: `false`

`true` Enable the use of hash symbols (`#`) as ordered item list markers:

```
#. Bird
#. McHale
#. Parish
```

`false` Disable the use of hash symbols (`#`) as ordered item list markers.

```
\@@_add_lua_option:nnn
  { hashEnumerators }
  { boolean }
  { false }
defaultOptions.hashEnumerators = false
```

`headerAttributes`=true, false default: `false`

`true` Enable the assignment of HTML attributes to headings:

```
# My first heading {#foo}

## My second heading ##    {#bar .baz}

Yet another heading   {key=value}
==================
```

These HTML attributes have currently no effect other than enabling content slicing, see the `slice` option.

`false` Disable the assignment of HTML attributes to headings.

```
\@@_add_lua_option:nnn
  { headerAttributes }
  { boolean }
  { false }
defaultOptions.headerAttributes = false
```

`html`=true, false default: `false`

`true` Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.

`false` Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```
\@@_add_lua_option:nnn
  { html }
  { boolean }
  { false }
defaultOptions.html = false
```

`hybrid`=true, false default: `false`

`true` Disable the escaping of special plain TEX characters, which makes it possible to intersperse your markdown markup with TEX code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix TEX and markdown markup freely.

`false` Enable the escaping of special plain TEX characters outside verbatim environments, so that they are not interpretted by TEX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
\@@_add_lua_option:nnn
  { hybrid }
  { boolean }
  { false }
defaultOptions.hybrid = false
```

`inlineFootnotes`=true, false default: `false`

`true` Enable the Pandoc inline footnote syntax extension:

```
Here is an inline note.^[Inlines notes are easier to
write, since you don't have to pick an identifier and
move down to type the note.]
```

`false` Disable the Pandoc inline footnote syntax extension.

```
\@@_add_lua_option:nnn
  { inlineFootnotes }
  { boolean }
  { false }
defaultOptions.inlineFootnotes = false
```

`jekyllData`=true, false default: `false`

`true` Enable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML:

```
---
title:  'This is the title: it contains a colon'
author:
- Author One
```

```
- Author Two
keywords: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
---
```

`false` Disable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML.

```
\@@_add_lua_option:nnn
  { jekyllData }
  { boolean }
  { false }
defaultOptions.jekyllData = false
```

`pipeTables`=true, `false` default: `false`

`true` Enable the PHP Markdown pipe table syntax extension:

```
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |
```

`false` Disable the PHP Markdown pipe table syntax extension.

```
\@@_add_lua_option:nnn
  { pipeTables }
  { boolean }
  { false }
defaultOptions.pipeTables = false
```

`preserveTabs`=true, `false` default: `false`

`true` Preserve tabs in code block and fenced code blocks.

`false` Convert any tabs in the input to spaces.

```
\@@_add_lua_option:nnn
  { preserveTabs }
  { boolean }
  { false }
defaultOptions.preserveTabs = false
```

`relativeReferences`=`true`, `false` default: `false`

`true` Enable relative references[6] in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
==========
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
```

`false` Disable relative references in autolinks.

```
\@@_add_lua_option:nnn
  { relativeReferences }
  { boolean }
  { false }
defaultOptions.relativeReferences = false
```

`shiftHeadings`=⟨*shift amount*⟩ default: `0`

All headings will be shifted by ⟨*shift amount*⟩, which can be both positive and negative. Headings will not be shifted beyond level 6 or below level 1. Instead, those headings will be shifted to level 6, when ⟨*shift amount*⟩ is positive, and to level 1, when ⟨*shift amount*⟩ is negative.

```
\@@_add_lua_option:nnn
  { shiftHeadings }
  { number }
  { 0 }
defaultOptions.shiftHeadings = 0
```

`slice`=⟨*the beginning and the end of a slice*⟩ default: `^ $`

Two space-separated selectors that specify the slice of a document that will be processed, whereas the remainder of the document will be ignored. The following selectors are recognized:

- The circumflex (`^`) selects the beginning of a document.
- The dollar sign (`$`) selects the end of a document.
- `^`⟨*identifier*⟩ selects the beginning of a section with the HTML attribute `#`⟨*identifier*⟩ (see the `headerAttributes` option).

[6] See https://datatracker.ietf.org/doc/html/rfc3986#section-4.2.

- `$`⟨*identifier*⟩ selects the end of a section with the HTML attribute `#`⟨*identifier*⟩.
- ⟨*identifier*⟩ corresponds to `^`⟨*identifier*⟩ for the first selector and to `$`⟨*identifier*⟩ for the second selector.

Specifying only a single selector, ⟨*identifier*⟩, is equivalent to specifying the two selectors ⟨*identifier*⟩ ⟨*identifier*⟩, which is equivalent to `^`⟨*identifier*⟩ `$`⟨*identifier*⟩, i.e. the entire section with the HTML attribute `#`⟨*identifier*⟩ will be selected.

```
\@@_add_lua_option:nnn
  { slice }
  { slice }
  { ^~$ }
defaultOptions.slice = "^ $"
```

`smartEllipses`=`true`, `false` default: `false`

`true` Convert any ellipses in the input to the `\markdownRendererEllipsis` TEX macro.

`false` Preserve all ellipses in the input.

```
\@@_add_lua_option:nnn
  { smartEllipses }
  { boolean }
  { false }
defaultOptions.smartEllipses = false
```

`startNumber`=`true`, `false` default: `true`

`true` Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererOlItemWithNumber` TEX macro.

`false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOlItem` TEX macro.

```
\@@_add_lua_option:nnn
  { startNumber }
  { boolean }
  { true }
defaultOptions.startNumber = true
```

`strikeThrough`=true, false default: `false`

`true` Enable the Pandoc strike-through syntax extension:

```
This ~~is deleted text.~~
```

`false` Disable the Pandoc strike-through syntax extension.

```
\@@_add_lua_option:nnn
  { strikeThrough }
  { boolean }
  { false }
defaultOptions.strikeThrough = false
```

`stripIndent`=true, false default: `false`

`true` Strip the minimal indentation of non-blank lines from all lines in a markdown document. Requires that the `preserveTabs` Lua option is disabled:

```
\documentclass{article}
\usepackage[stripIndent]{markdown}
\begin{document}
    \begin{markdown}
        Hello *world*!
    \end{markdown}
\end{document}
```

`false` Do not strip any indentation from the lines in a markdown document.

```
\@@_add_lua_option:nnn
  { stripIndent }
  { boolean }
  { false }
defaultOptions.stripIndent = false
```

`subscripts`=true, false default: `false`

`true` Enable the Pandoc subscript syntax extension:

```
H~2~O is a liquid.
```

`false` Disable the Pandoc subscript syntax extension.

```
\@@_add_lua_option:nnn
  { subscripts }
  { boolean }
  { false }
defaultOptions.subscripts = false
```

`superscripts`=true, `false` default: `false`

`true` Enable the Pandoc superscript syntax extension:

```
2^10^ is 1024.
```

`false` Disable the Pandoc superscript syntax extension.

```
\@@_add_lua_option:nnn
  { superscripts }
  { boolean }
  { false }
defaultOptions.superscripts = false
```

`tableCaptions`=true, `false` default: `false`

`true` Enable the Pandoc `table_captions` syntax extension for pipe tables (see the `pipeTables` option).

```
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

  : Demonstration of pipe table syntax.
```

`false` Disable the Pandoc `table_captions` syntax extension.

```
\@@_add_lua_option:nnn
  { tableCaptions }
  { boolean }
  { false }
defaultOptions.tableCaptions = false
```

`taskLists`=`true`, `false` default: `false`

`true` Enable the Pandoc `task_lists` syntax extension.

```
- [ ] an unticked task list item
- [/] a half-checked task list item
- [X] a ticked task list item
```

`false` Disable the Pandoc `task_lists` syntax extension.

```
\@@_add_lua_option:nnn
  { taskLists }
  { boolean }
  { false }
defaultOptions.taskLists = false
```

`texComments`=`true`, `false` default: `false`

`true` Strip TeX-style comments.

```
\documentclass{article}
\usepackage[texComments]{markdown}
\begin{document}
\begin{markdown}
Hello *world*!
\end{markdown}
\end{document}
```

Always enabled when `hybrid` is enabled.

`false` Do not strip TeX-style comments.

```
\@@_add_lua_option:nnn
  { texComments }
  { boolean }
  { false }
defaultOptions.texComments = false
```

`tightLists`=`true`, `false` default: `true`

`true` Unordered and ordered lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renderers that may produce different output than lists that are not tight:

```
- This is
- a tight
- unordered list.

- This is

  not a tight

- unordered list.
```

`false` Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

```
\@@_add_lua_option:nnn
  { tightLists }
  { boolean }
  { true }
defaultOptions.tightLists = true
```

`underscores`=`true`, `false` default: `true`

`true` Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```
*single asterisks*
_single underscores_
**double asterisks**
__double underscores__
```

`false` Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the `hybrid` option without the need to constantly escape subscripts.

```
\@@_add_lua_option:nnn
  { underscores }
  { boolean }
  { true }
\ExplSyntaxOff
defaultOptions.underscores = true
```

2.1.6 Command-Line Interface

The high-level operation of the Markdown package involves the communication between several programming layers: the plain TEX layer hands markdown documents to the Lua layer. Lua converts the documents to TEX, and hands the converted documents back to plain TEX layer for typesetting, see Figure 2.

This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted TEX documents are cached on the file system, taking up increasing amount of space. Unless the TEX engine includes a Lua interpreter, the package also requires shell access, which opens the door for a malicious actor to access the system. Last, but not least, the complexity of the procedure impedes debugging.

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to TEX is also provided, see Figure 3.

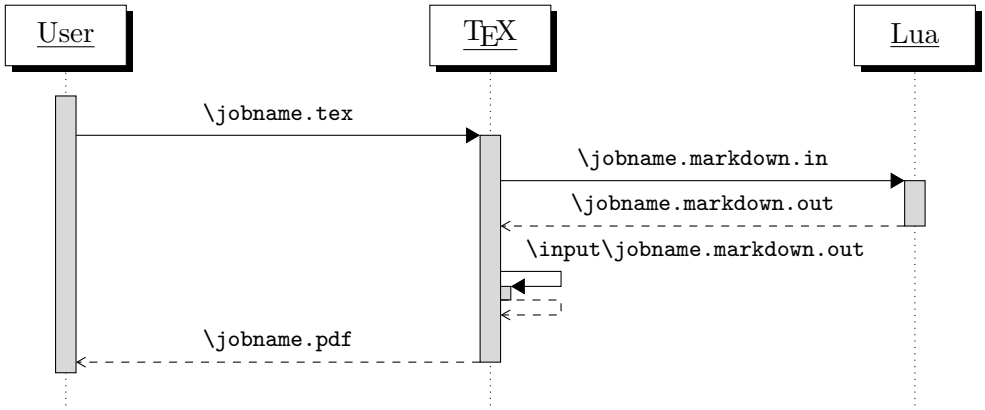

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
511
512 local HELP_STRING = [[
513 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
514 where OPTIONS are documented in the Lua interface section of the
515 technical Markdown package documentation.
516
517 When OUTPUT_FILE is unspecified, the result of the conversion will be
518 written to the standard output. When INPUT_FILE is also unspecified, the
519 result of the conversion will be read from the standard input.
520
521 Report bugs to: witiko@mail.muni.cz
522 Markdown package home page: <https://github.com/witiko/markdown>]]
523
```

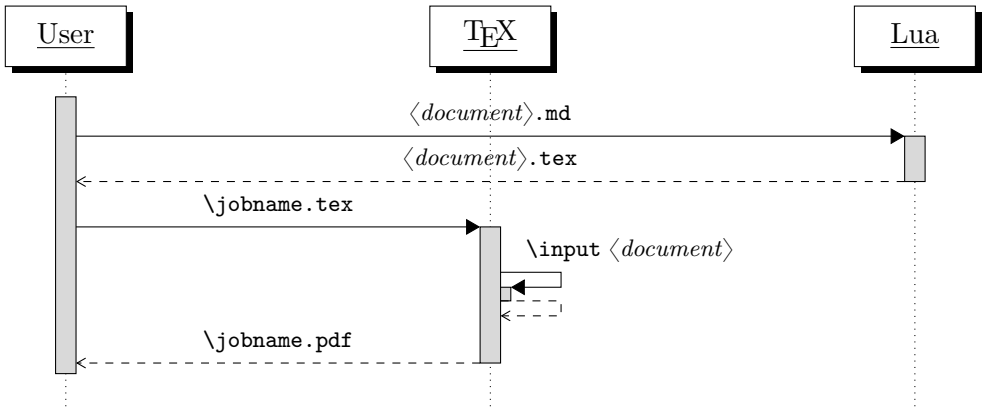

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
local VERSION_STRING = [[
markdown-cli.lua (Markdown) ]] .. metadata.version .. [[

Copyright (C) ]] .. table.concat(metadata.copyright,
                                 "\nCopyright (C) ") .. [[

License: ]] .. metadata.license

local function warn(s)
  io.stderr:write("Warning: " .. s .. "\n") end

local function error(s)
  io.stderr:write("Error: " .. s .. "\n")
  os.exit(1) end

local process_options = true
local options = {}
local input_filename
local output_filename
for i = 1, #arg do
  if process_options then
```

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

```
    if arg[i] == "--" then
      process_options = false
      goto continue
```

Unless the `--` argument has been specified before, an argument containing the equals sign (`=`) is assumed to be an option specification in a ⟨*key*⟩`=`⟨*value*⟩ format. The available options are listed in Section 2.1.3.

```
    elseif arg[i]:match("=") then
      local key, value = arg[i]:match("(.-)=(.*)")
```

The `defaultOptions` table is consulted to identify whether ⟨*value*⟩ should be parsed as a string or as a boolean.

```
      local default_type = type(defaultOptions[key])
      if default_type == "boolean" then
        options[key] = (value == "true")
      elseif default_type == "number" then
        options[key] = tonumber(value)
      elseif default_type == "table" then
        options[key] = {}
        for item in value:gmatch("[^ ,]+") do
          table.insert(options[key], item)
        end
      else
        if default_type ~= "string" then
          if default_type == "nil" then
            warn('Option "' .. key .. '" not recognized.')
          else
            warn('Option "' .. key .. '" type not recognized, please file ' ..
                 'a report to the package maintainer.')
          end
          warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
               key .. '" as a string.')
        end
        options[key] = value
      end
      goto continue
```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

```
    elseif arg[i] == "--help" or arg[i] == "-h" then
      print(HELP_STRING)
      os.exit()
```

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

```
    elseif arg[i] == "--version" or arg[i] == "-v" then
      print(VERSION_STRING)
      os.exit()
    end
```

```
581   end
```

The first argument that matches none of the above patters is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a TEX document.

```
582   if input_filename == nil then
583     input_filename = arg[i]
```

The first argument that matches none of the above patters is assumed to be the output filename. The output filename should correspond to the TEX document that will result from the conversion.

```
584   elseif output_filename == nil then
585     output_filename = arg[i]
586   else
587     error('Unexpected argument: "' .. arg[i] .. '".')
588   end
589   ::continue::
590 end
```

The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```
texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex
```

to convert the Markdown document `hello.md` to a TEX document `hello.tex`. After the Markdown package for our TEX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

2.2 Plain TEX Interface

The plain TEX interface provides macros for the typesetting of markdown input from within plain TEX, for setting the Lua interface options (see Section 2.1.3) used during the conversion from markdown to plain TEX and for changing the way markdown the tokens are rendered.

```
591 \def\markdownLastModified{(((LASTMODIFIED)))}%
592 \def\markdownVersion{(((VERSION)))}%
```

The plain TEX interface is implemented by the `markdown.tex` file that can be loaded as follows:

```
\input markdown
```

It is expected that the special plain TEX characters have the expected category codes, when `\input`ting the file.

2.2.1 Typesetting Markdown

The interface exposes the \markdownBegin, \markdownEnd, and \markdownInput macros.

The \markdownBegin macro marks the beginning of a markdown document fragment and the \markdownEnd macro marks its end.

```
593 \let\markdownBegin\relax
594 \let\markdownEnd\relax
```

You may prepend your own code to the \markdownBegin macro and redefine the \markdownEnd macro to produce special effects before and after the markdown block.

There are several limitations to the macros you need to be aware of. The first limitation concerns the \markdownEnd macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corrolary, the \markdownEnd string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of TEX [4, p. 46]. As a corrolary, the \markdownBegin macro also ignores them.

The \markdownBegin and \markdownEnd macros will also consume the rest of the lines at which they appear. In the following example plain TEX code, the characters c, e, and f will not appear in the output.

```
\input markdown
a
b \markdownBegin c
d
e \markdownEnd   f
g
\bye
```

Note that you may also not nest the \markdownBegin and \markdownEnd macros.

The following example plain TEX code showcases the usage of the \markdownBegin and \markdownEnd macros:

```
\input markdown
\markdownBegin
_Hello_ **world** ...
\markdownEnd
\bye
```

The \markdownInput macro accepts a single parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TEX.

```
\let\markdownInput\relax
```

This macro is not subject to the abovelisted limitations of the \markdownBegin and \markdownEnd macros.

The following example plain TEX code showcases the usage of the \markdownInput macro:

```
\input markdown
\markdownInput{hello.md}
\bye
```

2.2.2 Options

The plain TEX options are represented by TEX commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain TEX interface.

To enable the enumeration of plain TEX options, we will maintain the \g_@@_plain_tex_options_seq sequence.

```
\ExplSyntaxOn
\seq_new:N \g_@@_plain_tex_options_seq
```

To enable the reflection of default plain TEX options and their types, we will maintain the \g_@@_default_plain_tex_options_prop and \g_@@_plain_tex_option_types_prop property lists, respectively.

```
\prop_new:N \g_@@_plain_tex_option_types_prop
\prop_new:N \g_@@_default_plain_tex_options_prop
\tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
\seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
\cs_new:Nn
  \@@_add_plain_tex_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_plain_tex_tl
      { #1 }
      { #2 }
      { #3 }
  }
```

2.2.2.1 Finalizing and Freezing the Cache

The \markdownOptionFinalizeCache option corresponds to the Lua interface finalizeCache option, which creates an output file \markdownOptionFrozenCacheFileName (frozen cache) that contains a

mapping between an enumeration of the markdown documents in the plain TEX document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `\markdownOptionFinalizeCache` option, and uses it to typeset the plain TEX document without invoking Lua. As a result, the plain TEX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

```
\@@_add_plain_tex_option:nnn
  { frozenCache }
  { boolean }
  { false }
```

The standard usage of the above two options is as follows:

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `\markdownOptionFinalizeCache` option.
4. Typeset the plain TEX document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain TEX document and the `cacheDir` directory.

2.2.2.2 File and Directory Names

The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain TEX in TEX engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

The expansion of this macro must not contain quotation marks (`"`) or backslash symbols (`\`). Mind that TEX engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
\@@_add_plain_tex_option:nnn
  { helperScriptFileName }
  { path }
  { \jobname.markdown.lua }
```

The `\markdownOptionHelperScriptFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the helper Lua script file, use the `\g_luabridge_helper_script_filename_str` macro from the lt3luabridge package.

```
\str_new:N
  \g_luabridge_helper_script_filename_str
\tl_gset:Nn
  \g_luabridge_helper_script_filename_str
  { \markdownOptionHelperScriptFileName }
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a TEX source.

It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```
\@@_add_plain_tex_option:nnn
  { inputTempFileName }
  { path }
  { \jobname.markdown.in }
```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain TEX in `\markdownMode` other than `2` It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
\@@_add_plain_tex_option:nnn
  { outputTempFileName }
  { path }
  { \jobname.markdown.out }
```

The `\markdownOptionOutputTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0.

```
\str_new:N
  \g_luabridge_standard_output_filename_str
\tl_gset:Nn
  \g_luabridge_standard_output_filename_str
  { \markdownOptionOutputTempFileName }
```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain TEX in `\markdownMode` other than `2`. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
\@@_add_plain_tex_option:nnn
  { errorTempFileName }
  { path }
  { \jobname.markdown.err }
```

The `\markdownOptionErrorTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the temporary file for Lua errors, use the `\g_luabridge_error_output_filename_str` macro from the lt3luabridge package.

```
\str_new:N
  \g_luabridge_error_output_filename_str
\tl_gset:Nn
  \g_luabridge_error_output_filename_str
  { \markdownOptionErrorTempFileName }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain TEX implementation. The option defaults to `..`

The path must be set to the same value as the `-output-directory` option of your TEX engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
\@@_add_plain_tex_option:nnn
  { outputDir }
  { path }
  { . }
```

Here, we automatically define plain TEX macros for the above plain TEX options.

Furthemore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain TEX implementation, only passed along to Lua.

For the macros that correspond to the non-boolean options recognized by the Lua interface, the same limitations apply here in the case of the `\markdownOptionHelperScriptFileName` macro.

```
\cs_new:Nn \@@_plain_tex_define_option_commands:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_plain_tex_define_option_command:n
                { ####1 }
          }
      }
  }
\cs_new:Nn \@@_plain_tex_define_option_command:n
  {
    \@@_get_default_option_value:nN
      { #1 }
      \l_tmpa_tl
    \@@_set_option_value:nV
      { #1 }
      \l_tmpa_tl
  }
\cs_new:Nn
  \@@_set_option_value:nn
  {
    \@@_define_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
```

```
      \l_tmpa_tl
    \str_if_eq:NNTF
      \c_@@_option_type_counter_tl
      \l_tmpa_tl
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \int_gset:cn
          { \l_tmpa_tl }
          { #2 }
      }
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_set:cpn
          { \l_tmpa_tl }
          { #2 }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\cs_new:Nn
  \@@_define_option:n
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cT
      { \l_tmpa_tl }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpb_tl
        \str_if_eq:NNT
          \c_@@_option_type_counter_tl
          \l_tmpb_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \int_new:c
              { \l_tmpa_tl }
          }
      }
```

```
  }
\@@_plain_tex_define_option_commands:
```

2.2.2.3 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (`%`) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing TEX package documentation using the Doc LATEX package [5] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
\seq_put_right:Nn
  \g_@@_plain_tex_options_seq
  { stripPercentSigns }
\prop_put:Nnn
  \g_@@_plain_tex_option_types_prop
  { stripPercentSigns }
  { boolean }
\prop_put:Nnx
  \g_@@_default_plain_tex_options_prop
  { stripPercentSigns }
  { false }
\ExplSyntaxOff
```

2.2.3 Token Renderers

The following TEX macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.4).

To enable the enumeration of token renderers, we will maintain the `\g_@@_renderers_seq` sequence.

```
\ExplSyntaxOn
\seq_new:N \g_@@_renderers_seq
```

To enable the reflection of token renderers and their parameters, we will maintain the `\g_@@_renderer_arities_prop` property list.

```
\prop_new:N \g_@@_renderer_arities_prop
\ExplSyntaxOff
```

2.2.3.1 Tickbox Renderers

The macros named `\markdownRendererTickedBox`, `\markdownRendererHalfTickedBox`, and `\markdownRendererUntickedBox` represent ticked and unticked boxes, respectively. These macros will either be produced, when the `taskLists` option is enabled, or when the Ballot Box with X (☒, U+2612), Hourglass (⊡, U+231B) or Ballot Box (▫, U+2610) Unicode characters are encountered in the markdown input, respectively.

```
\def\markdownRendererTickedBox{%
  \markdownRendererTickedBoxPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { tickedBox }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { tickedBox }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHalfTickedBox{%
  \markdownRendererHalfTickedBoxPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { halfTickedBox }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { halfTickedBox }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererUntickedBox{%
  \markdownRendererUntickedBoxPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { untickedBox }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { untickedBox }
  { 0 }
\ExplSyntaxOff
```

2.2.3.2 Markdown Document Renderers

The `\markdownRendererDocumentBegin` and `\markdownRendererDocumentEnd` macros represent the beginning and the end of a *markdown* document. The macros receive no arguments.

A TEX document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```
\def\markdownRendererDocumentBegin{%
  \markdownRendererDocumentBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
```

```
  \g_@@_renderers_seq
  { documentBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { documentBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDocumentEnd{%
  \markdownRendererDocumentEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { documentEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { documentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.3 Interblock Separator Renderer

The `\markdownRendererInterblockSeparator` macro represents a separator between two markdown block elements. The macro receives no arguments.

```
\def\markdownRendererInterblockSeparator{%
  \markdownRendererInterblockSeparatorPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { interblockSeparator }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { interblockSeparator }
  { 0 }
\ExplSyntaxOff
```

2.2.3.4 Line Break Renderer

The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

```
\def\markdownRendererLineBreak{%
  \markdownRendererLineBreakPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { lineBreak }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { lineBreak }
```

```
  { 0 }
\ExplSyntaxOff
```

2.2.3.5 Ellipsis Renderer

The `\markdownRendererEllipsis` macro replaces any occurance of ASCII ellipses in the input text. This macro will only be produced, when the `smartEllipses` option is enabled. The macro receives no arguments.

```
\def\markdownRendererEllipsis{%
  \markdownRendererEllipsisPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ellipsis }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

2.2.3.6 Non-Breaking Space Renderer

The `\markdownRendererNbsp` macro represents a non-breaking space.

```
\def\markdownRendererNbsp{%
  \markdownRendererNbspPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { nbsp }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { nbsp }
  { 0 }
\ExplSyntaxOff
```

2.2.3.7 Special Character Renderers

The following macros replace any special plain TEX characters, including the active pipe character (`|`) of ConTEXt, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```
\def\markdownRendererLeftBrace{%
  \markdownRendererLeftBracePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { leftBrace }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { leftBrace }
```

```
  { 0 }
\ExplSyntaxOff
\def\markdownRendererRightBrace{%
  \markdownRendererRightBracePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { rightBrace }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { rightBrace }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDollarSign{%
  \markdownRendererDollarSignPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dollarSign }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dollarSign }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPercentSign{%
  \markdownRendererPercentSignPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { percentSign }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { percentSign }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererAmpersand{%
  \markdownRendererAmpersandPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ampersand }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ampersand }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererUnderscore{%
```

```
  \markdownRendererUnderscorePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { underscore }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { underscore }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHash{%
  \markdownRendererHashPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { hash }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { hash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererCircumflex{%
  \markdownRendererCircumflexPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { circumflex }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { circumflex }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBackslash{%
  \markdownRendererBackslashPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { backslash }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { backslash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererTilde{%
  \markdownRendererTildePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
```

```
  \g_@@_renderers_seq
  { tilde }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { tilde }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPipe{%
  \markdownRendererPipePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { pipe }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { pipe }
  { 0 }
\ExplSyntaxOff
```

2.2.3.8 Code Span Renderer

The `\markdownRendererCodeSpan` macro represents inlined code span in the input text. It receives a single argument that corresponds to the inlined code span.

```
\def\markdownRendererCodeSpan{%
  \markdownRendererCodeSpanPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { codeSpan }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { codeSpan }
  { 1 }
\ExplSyntaxOff
```

2.2.3.9 Link Renderer

The `\markdownRendererLink` macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
\def\markdownRendererLink{%
  \markdownRendererLinkPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { link }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
```

```
  { link }
  { 4 }
\ExplSyntaxOff
```

2.2.3.10 Image Renderer

The `\markdownRendererImage` macro represents an image. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

```
\def\markdownRendererImage{%
  \markdownRendererImagePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { image }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { image }
  { 4 }
\ExplSyntaxOff
```

2.2.3.11 Content Block Renderers

The `\markdownRendererContentBlock` macro represents an iA,Writer content block. It receives four arguments: the local file or online image filename extension cast to the lower case, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

```
\def\markdownRendererContentBlock{%
  \markdownRendererContentBlockPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { contentBlock }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { contentBlock }
  { 4 }
\ExplSyntaxOff
```

The `\markdownRendererContentBlockOnlineImage` macro represents an iA,Writer online image content block. The macro receives the same arguments as `\markdownRendererContentBlock`.

```
\def\markdownRendererContentBlockOnlineImage{%
  \markdownRendererContentBlockOnlineImagePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
```

```
  \g_@@_renderers_seq
  { contentBlockOnlineImage }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockOnlineImage }
  { 4 }
\ExplSyntaxOff
```

The `\markdownRendererContentBlockCode` macro represents an iA,Writer content block that was recognized as a file in a known programming language by its filename extension s. If any `markdown-languages.json` file found by kpathsea[7] contains a record (k, v), then a non-online-image content block with the filename extension s, `s:lower()` $= k$ is considered to be in a known programming language v. The macro receives five arguments: the local file name extension s cast to the lower case, the language v, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place place a `markdown-languages.json` file inside your working directory or inside your local TEX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```
\def\markdownRendererContentBlockCode{%
  \markdownRendererContentBlockCodePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

2.2.3.12 Bullet List Renderers

The `\markdownRendererUlBegin` macro represents the beginning of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
\def\markdownRendererUlBegin{%
  \markdownRendererUlBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
```

[7] Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```
  { ulBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ulBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererUlBeginTight` macro represents the beginning of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```
\def\markdownRendererUlBeginTight{%
  \markdownRendererUlBeginTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ulBeginTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ulBeginTight }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererUlItem` macro represents an item in a bulleted list. The macro receives no arguments.

```
\def\markdownRendererUlItem{%
  \markdownRendererUlItemPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ulItem }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ulItem }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererUlItemEnd` macro represents the end of an item in a bulleted list. The macro receives no arguments.

```
\def\markdownRendererUlItemEnd{%
  \markdownRendererUlItemEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ulItemEnd }
\prop_put:Nnn
```

```
  \g_@@_renderer_arities_prop
  { ulItemEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererUlEnd macro represents the end of a bulleted list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
\def\markdownRendererUlEnd{%
  \markdownRendererUlEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ulEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ulEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererUlEndTight macro represents the end of a bulleted list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the tightLists option is disabled. The macro receives no arguments.

```
\def\markdownRendererUlEndTight{%
  \markdownRendererUlEndTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { ulEndTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { ulEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.13 Ordered List Renderers

The \markdownRendererOlBegin macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the fancyLists option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlBegin{%
  \markdownRendererOlBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
```

```
  { olBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererOlBeginTight` macro represents the beginning of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlBeginTight{%
  \markdownRendererOlBeginTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olBeginTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olBeginTight }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererFancyOlBegin` macro represents the beginning of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives two arguments: the style of the list item labels (`Decimal`, `LowerRoman`, `UpperRoman`, `LowerAlpha`, and `UpperAlpha`), and the style of delimiters between list item labels and texts (`Default`, `OneParen`, and `Period`).

```
\def\markdownRendererFancyOlBegin{%
  \markdownRendererFancyOlBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBegin }
  { 2 }
\ExplSyntaxOff
```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item

labels, and the style of delimiters between list item labels and texts. See the \markdownRendererFancyOlBegin macro for the valid style values.

```
\def\markdownRendererFancyOlBeginTight{%
  \markdownRendererFancyOlBeginTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlBeginTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBeginTight }
  { 2 }
\ExplSyntaxOff
```

The \markdownRendererOlItem macro represents an item in an ordered list. This macro will only be produced, when the startNumber option is disabled and the fancyLists option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlItem{%
  \markdownRendererOlItemPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olItem }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olItem }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererOlItemEnd macro represents the end of an item in an ordered list. This macro will only be produced, when the fancyLists option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlItemEnd{%
  \markdownRendererOlItemEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olItemEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olItemEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererOlItemWithNumber macro represents an item in an ordered list. This macro will only be produced, when the startNumber option is

enabled and the `fancyLists` option is disabled. The macro receives a single numeric argument that corresponds to the item number.

```
\def\markdownRendererOlItemWithNumber{%
  \markdownRendererOlItemWithNumberPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olItemWithNumber }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererFancyOlItem` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```
\def\markdownRendererFancyOlItem{%
  \markdownRendererFancyOlItemPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlItem }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItem }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererFancyOlItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```
\def\markdownRendererFancyOlItemEnd{%
  \markdownRendererFancyOlItemEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererFancyOlItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and

`fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```
\def\markdownRendererFancyOlItemWithNumber{%
  \markdownRendererFancyOlItemWithNumberPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemWithNumber }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererOlEnd` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlEnd{%
  \markdownRendererOlEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererOlEndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```
\def\markdownRendererOlEndTight{%
  \markdownRendererOlEndTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { olEndTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { olEndTight }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererFancyOlEnd macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the fancyLists option is enabled. The macro receives no arguments.

```
\def\markdownRendererFancyOlEnd{%
  \markdownRendererFancyOlEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererFancyOlEndTight macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the fancyLists and tightLists options are enabled. The macro receives no arguments.

```
\def\markdownRendererFancyOlEndTight{%
  \markdownRendererFancyOlEndTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { fancyOlEndTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.14 Definition List Renderers

The following macros are only produced, when the definitionLists option is enabled.

The \markdownRendererDlBegin macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
\def\markdownRendererDlBegin{%
  \markdownRendererDlBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlBegin }
\prop_put:Nnn
```

```
  \g_@@_renderer_arities_prop
  { dlBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```
\def\markdownRendererDlBeginTight{%
  \markdownRendererDlBeginTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlBeginTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlBeginTight }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererDlItem` macro represents a term in a definition list. The macro receives a single argument that corresponds to the term being defined.

```
\def\markdownRendererDlItem{%
  \markdownRendererDlItemPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlItem }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlItem }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererDlItemEnd` macro represents the end of a list of definitions for a single term.

```
\def\markdownRendererDlItemEnd{%
  \markdownRendererDlItemEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlItemEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlItemEnd }
```

```
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererDlDefinitionBegin` macro represents the beginning of a definition in a definition list. There can be several definitions for a single term.

```
\def\markdownRendererDlDefinitionBegin{%
  \markdownRendererDlDefinitionBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererDlDefinitionEnd` macro represents the end of a definition in a definition list. There can be several definitions for a single term.

```
\def\markdownRendererDlDefinitionEnd{%
  \markdownRendererDlDefinitionEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererDlEnd` macro represents the end of a definition list that contains an item with several paragraphs of text (the list is not tight). The macro receives no arguments.

```
\def\markdownRendererDlEnd{%
  \markdownRendererDlEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererDlEndTight macro represents the end of a definition list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the tightLists option is disabled. The macro receives no arguments.

```
\def\markdownRendererDlEndTight{%
  \markdownRendererDlEndTightPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { dlEndTight }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { dlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.15 Emphasis Renderers

The \markdownRendererEmphasis macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```
\def\markdownRendererEmphasis{%
  \markdownRendererEmphasisPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { emphasis }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { emphasis }
  { 1 }
\ExplSyntaxOff
```

The \markdownRendererStrongEmphasis macro represents a strongly emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```
\def\markdownRendererStrongEmphasis{%
  \markdownRendererStrongEmphasisPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { strongEmphasis }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { strongEmphasis }
  { 1 }
\ExplSyntaxOff
```

2.2.3.16 Block Quote Renderers

The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```
\def\markdownRendererBlockQuoteBegin{%
  \markdownRendererBlockQuoteBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { blockQuoteBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererBlockQuoteEnd` macro represents the end of a block quote. The macro receives no arguments.

```
\def\markdownRendererBlockQuoteEnd{%
  \markdownRendererBlockQuoteEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { blockQuoteEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.17 Code Block Renderers

The `\markdownRendererInputVerbatim` macro represents a code block. The macro receives a single argument that corresponds to the filename of a file contaning the code block contents.

```
\def\markdownRendererInputVerbatim{%
  \markdownRendererInputVerbatimPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { inputVerbatim }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { inputVerbatim }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The

macro receives two arguments that correspond to the filename of a file contaning the code block contents and to the code fence infostring.

```
\def\markdownRendererInputFencedCode{%
  \markdownRendererInputFencedCodePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { inputFencedCode }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { inputFencedCode }
  { 2 }
\ExplSyntaxOff
```

2.2.3.18 YAML Metadata Renderers

The `\markdownRendererJekyllDataBegin` macro represents the beginning of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```
\def\markdownRendererJekyllDataBegin{%
  \markdownRendererJekyllDataBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBegin }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataEnd` macro represents the end of a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives no arguments.

```
\def\markdownRendererJekyllDataEnd{%
  \markdownRendererJekyllDataEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the

jekyllData option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the mapping.

```
\def\markdownRendererJekyllDataMappingBegin{%
  \markdownRendererJekyllDataMappingBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingBegin }
  { 2 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataMappingEnd macro represents the end of a mapping in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives no arguments.

```
\def\markdownRendererJekyllDataMappingEnd{%
  \markdownRendererJekyllDataMappingEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataSequenceBegin macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the sequence.

```
\def\markdownRendererJekyllDataSequenceBegin{%
  \markdownRendererJekyllDataSequenceBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceBegin }
  { 2 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataSequenceEnd macro represents the end of a sequence in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives no arguments.

```
\def\markdownRendererJekyllDataSequenceEnd{%
  \markdownRendererJekyllDataSequenceEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataBoolean macro represents a boolean scalar value in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```
\def\markdownRendererJekyllDataBoolean{%
  \markdownRendererJekyllDataBooleanPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBoolean }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBoolean }
  { 2 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataNumber macro represents a numeric scalar value in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives two arguments: the scalar key in the parent structure, and the scalar value, both cast to a string following YAML serialization rules.

```
\def\markdownRendererJekyllDataNumber{%
  \markdownRendererJekyllDataNumberPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataNumber }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
```

```
  { jekyllDataNumber }
  { 2 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```
\def\markdownRendererJekyllDataString{%
  \markdownRendererJekyllDataStringPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataString }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataString }
  { 2 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataEmpty` macro represents an empty scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives one argument: the scalar key in the parent structure, cast to a string following YAML serialization rules.

See also Section 2.2.4.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```
\def\markdownRendererJekyllDataEmpty{%
  \markdownRendererJekyllDataEmptyPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEmpty }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEmpty }
  { 1 }
\ExplSyntaxOff
```

2.2.3.19 Heading Renderers

The `\markdownRendererHeadingOne` macro represents a first level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingOne{%
  \markdownRendererHeadingOnePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
```

```
  \g_@@_renderers_seq
  { headingOne }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingOne }
  { 1 }
\ExplSyntaxOff
```

The \markdownRendererHeadingTwo macro represents a second level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingTwo{%
  \markdownRendererHeadingTwoPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headingTwo }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingTwo }
  { 1 }
\ExplSyntaxOff
```

The \markdownRendererHeadingThree macro represents a third level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingThree{%
  \markdownRendererHeadingThreePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headingThree }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingThree }
  { 1 }
\ExplSyntaxOff
```

The \markdownRendererHeadingFour macro represents a fourth level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingFour{%
  \markdownRendererHeadingFourPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headingFour }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingFour }
```

```
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererHeadingFive` macro represents a fifth level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingFive{%
  \markdownRendererHeadingFivePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headingFive }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingFive }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererHeadingSix` macro represents a sixth level heading. The macro receives a single argument that corresponds to the heading text.

```
\def\markdownRendererHeadingSix{%
  \markdownRendererHeadingSixPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headingSix }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headingSix }
  { 1 }
\ExplSyntaxOff
```

2.2.3.20 Horizontal Rule Renderer

The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

```
\def\markdownRendererHorizontalRule{%
  \markdownRendererHorizontalRulePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { horizontalRule }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { horizontalRule }
  { 0 }
\ExplSyntaxOff
```

2.2.3.21 Footnote Renderer

The `\markdownRendererFootnote` macro represents a footnote. This macro will only be produced, when the `footnotes` option is enabled. The macro receives a single argument that corresponds to the footnote text.

```
\def\markdownRendererFootnote{%
  \markdownRendererFootnotePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { footnote }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { footnote }
  { 1 }
\ExplSyntaxOff
```

2.2.3.22 Parenthesized Citations Renderer

The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the `citations` option is enabled. The macro receives the parameter `{`⟨*number of citations*⟩`}` followed by ⟨*suppress author*⟩ `{`⟨*prenote*⟩`}{`⟨*postnote*⟩`}{`⟨*name*⟩`}` repeated ⟨*number of citations*⟩ times. The ⟨*suppress author*⟩ parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```
\def\markdownRendererCite{%
  \markdownRendererCitePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { cite }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { cite }
  { 1 }
\ExplSyntaxOff
```

2.2.3.23 Text Citations Renderer

The `\markdownRendererTextCite` macro represents a string of one or more text citations. This macro will only be produced, when the `citations` option is enabled. The macro receives parameters in the same format as the `\markdownRendererCite` macro.

```
\def\markdownRendererTextCite{%
  \markdownRendererTextCitePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
```

```
  { textCite }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { textCite }
  { 1 }
\ExplSyntaxOff
```

2.2.3.24 Table Renderer

The `\markdownRendererTable` macro represents a table. This macro will only be produced, when the `pipeTables` option is enabled. The macro receives the parameters {⟨*caption*⟩}{⟨*number of rows*⟩}{⟨*number of columns*⟩} followed by {⟨*alignments*⟩} and then by {⟨*row*⟩} repeated ⟨*number of rows*⟩ times, where ⟨*row*⟩ is {⟨*column*⟩} repeated ⟨*number of columns*⟩ times, ⟨*alignments*⟩ is ⟨*alignment*⟩ repeated ⟨*number of columns*⟩ times, and ⟨*alignment*⟩ is one of the following:

- `d` – The corresponding column has an unspecified (default) alignment.
- `l` – The corresponding column is left-aligned.
- `c` – The corresponding column is centered.
- `r` – The corresponding column is right-aligned.

```
\def\markdownRendererTable{%
  \markdownRendererTablePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { table }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { table }
  { 3 }
\ExplSyntaxOff
```

2.2.3.25 HTML Comment Renderers

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

The `\markdownRendererBlockHtmlCommentBegin` and `\markdownRendererBlockHtmlCommentEn` macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```
\def\markdownRendererInlineHtmlComment{%
  \markdownRendererInlineHtmlCommentPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlComment }
```

```
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlComment }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererBlockHtmlCommentBegin{%
  \markdownRendererBlockHtmlCommentBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { blockHtmlCommentBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlCommentBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBlockHtmlCommentEnd{%
  \markdownRendererBlockHtmlCommentEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { blockHtmlCommentEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlCommentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.26 HTML Tag and Element Renderers

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

The `\markdownRendererInputBlockHtmlElement` macro represents a block HTML element. This macro will only be produced, when the `html` option is enabled. The macro receives a single argument that filename of a file containing the contents of the HTML element.

```
\def\markdownRendererInlineHtmlTag{%
  \markdownRendererInlineHtmlTagPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlTag }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlTag }
```

```
  { 1 }
\ExplSyntaxOff
\def\markdownRendererInputBlockHtmlElement{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlElement }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlElement }
  { 1 }
\ExplSyntaxOff
```

2.2.3.27 Attribute Renderers

The following macros are only produced, when the `headerAttributes` option is enabled.

`\markdownRendererAttributeIdentifier` represents the ⟨*identifier*⟩ of a markdown element (`id="`⟨*identifier*⟩`"` in HTML and `#`⟨*identifier*⟩ in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the ⟨*identifier*⟩.

`\markdownRendererAttributeClassName` represents the ⟨*class name*⟩ of a markdown element (`class="`⟨*class name*⟩ ...`"` in HTML and `.`⟨*class name*⟩ in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the ⟨*class name*⟩.

`\markdownRendererAttributeKeyValue` represents a HTML attribute in the form ⟨*key*⟩`=`⟨*value*⟩ that is neither an identifier nor a class name. The macro receives two attributes that correspond to the ⟨*key*⟩ and the ⟨*value*⟩, respectively.

```
\def\markdownRendererAttributeIdentifier{%
  \markdownRendererAttributeIdentifierPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { attributeIdentifier }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { attributeIdentifier }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererAttributeClassName{%
  \markdownRendererAttributeClassNamePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { attributeClassName }
\prop_put:Nnn
```

```
  \g_@@_renderer_arities_prop
  { attributeClassName }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererAttributeKeyValue{%
  \markdownRendererAttributeKeyValuePrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { attributeKeyValue }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { attributeKeyValue }
  { 2 }
\ExplSyntaxOff
```

2.2.3.28 Header Attribute Context Renderers

The following macros are only produced, when the `headerAttributes` option is enabled.

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttri` macros represent the beginning and the end of a section in which the attributes of a heading apply. The macros receive no arguments.

```
\def\markdownRendererHeaderAttributeContextBegin{%
  \markdownRendererHeaderAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextBegin }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHeaderAttributeContextEnd{%
  \markdownRendererHeaderAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextEnd }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.29 Strike-Through Renderer

The `\markdownRendererStrikeThrough` macro represents a strike-through span of text. The macro receives a single argument that corresponds to the striked-out span of text. This macro will only be produced, when the `strikeThrough` option is enabled.

```
\def\markdownRendererStrikeThrough{%
  \markdownRendererStrikeThroughPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { strikeThrough }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { strikeThrough }
  { 1 }
\ExplSyntaxOff
```

2.2.3.30 Superscript Renderer

The `\markdownRendererSuperscript` macro represents a superscript span of text. The macro receives a single argument that corresponds to the superscript span of text. This macro will only be produced, when the `superscripts` option is enabled.

```
\def\markdownRendererSuperscript{%
  \markdownRendererSuperscriptPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { superscript }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
  { superscript }
  { 1 }
\ExplSyntaxOff
```

2.2.3.31 Subscript Renderer

The `\markdownRendererSubscript` macro represents a subscript span of text. The macro receives a single argument that corresponds to the subscript span of text. This macro will only be produced, when the `subscripts` option is enabled.

```
\def\markdownRendererSubscript{%
  \markdownRendererSubscriptPrototype}%
\ExplSyntaxOn
\seq_put_right:Nn
  \g_@@_renderers_seq
  { subscript }
\prop_put:Nnn
  \g_@@_renderer_arities_prop
```

```
  { subscript }
  { 1 }
\ExplSyntaxOff
```

2.2.4 Token Renderer Prototypes

2.2.4.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key–values from the l3keys module of the LaTeX3 kernel.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  { }
\ExplSyntaxOff
```

The following TeX macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the LaTeX and ConTeXt implementations (see sections 3.3 and 3.4).

```
\ExplSyntaxOn
\cs_new:Nn \@@_plaintex_define_renderer_prototypes:
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_plaintex_define_renderer_prototype:n
    \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
    \let\markdownRendererBlockHtmlCommentBegin=\iffalse
    \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
    \let\markdownRendererBlockHtmlCommentEnd=\fi
  }
\cs_new:Nn \@@_plaintex_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_plaintex_define_renderer_prototype:cV
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
  {
```

```
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
        Prototype
      }
  }
\cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
  {
    \cs_generate_from_arg_count:NNnn
      #1
      \cs_set:Npn
      { #2 }
      { }
  }
\cs_generate_variant:Nn
  \@@_plaintex_define_renderer_prototype:Nn
  { cV }
\@@_plaintex_define_renderer_prototypes:
\ExplSyntaxOff
```

2.2.5 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. You may redefine these macros to redirect and process the info, warning, and error messages.

2.2.6 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a TEX engine that does not support direct Lua access is starting to buffer a text. The plain TEX implementation changes the category code of plain TEX special characters to other, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
\let\markdownMakeOther\relax
```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain

TEX special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
\let\markdownReadAndConvert\relax
\begingroup
```

Locally swap the category code of the backslash symbol (`\`) with the pipe symbol (`|`). This is required in order that all the special symbols in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```
  \catcode`\|=0\catcode`\\=12%
  |gdef|markdownBegin{%
    |markdownReadAndConvert{\markdownEnd}%
                           {|markdownEnd}}%
|endgroup
```

The macro is exposed in the interface, so that the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.6), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

The `\markdownMode` macro specifies how the plain TEX implementation interfaces with the Lua interface. The valid values and their meaning are as follows:

- `0` – Shell escape via the 18 output file stream
- `1` – Shell escape via the Lua `os.execute` method
- `2` – Direct Lua access
- `3` – The lt3luabridge Lua package

By defining the macro, the user can coerce the package to use a specific mode. If the user does not define the macro prior to loading the plain TEX implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

The `\markdownMode` macro has been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
\ExplSyntaxOn
\cs_if_exist:NF
  \markdownMode
  {
    \file_if_exist:nTF
      { lt3luabridge.tex }
      {
        \cs_new:Npn
          \markdownMode
          { 3 }
      }
      {
        \cs_if_exist:NTF
```

```
            \directlua
            {
              \cs_new:Npn
                \markdownMode
                { 2 }
            }
            {
              \cs_new:Npn
                \markdownMode
                { 0 }
            }
        }
    }
```

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros have been deprecated and will be removed in Markdown 3.0.0:

```
\def\markdownLuaRegisterIBCallback#1{\relax}%
\def\markdownLuaUnregisterIBCallback#1{\relax}%
```

2.3 LaTeX Interface

The LaTeX interface provides LaTeX environments for the typesetting of markdown input from within LaTeX, facilities for setting Lua, plain TeX, and LaTeX options used during the conversion from markdown to plain TeX, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain TeX interface (see Section 2.2).

The LaTeX implementation redefines the plain TeX logging macros (see Section 3.2.1) to use the LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```
\newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
\newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
\newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
\input markdown/markdown
```

The LaTeX interface is implemented by the `markdown.sty` file, which can be loaded from the LaTeX document preamble as follows:

```
\usepackage[⟨options⟩]{markdown}
```

where ⟨*options*⟩ are the LaTeX interface options (see Section 2.3.2). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. This limitation is due to the way LaTeX 2$_\varepsilon$ parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` LaTeX environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` LaTeX environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts LaTeX interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
\newenvironment{markdown}\relax\relax
\newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` LaTeX environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` LaTeX environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain TeX interface.

The following example LaTeX code showcases the usage of the `markdown` and `markdown*` environments:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown}
_Hello_ **world** ...
\end{markdown}
% ...
\end{document}
```

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown*}{smartEllipses}
_Hello_ **world** ...
\end{markdown*}
% ...
\end{document}
```

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TeX. Unlike the `\markdownInput` macro provided by the plain TeX interface, this macro also accepts LaTeX interface options (see Section 2.3.2) as its optional argument. These options will only influnce this markdown document.

The following example LaTeX code showcases the usage of the `\markdownInput` macro:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[smartEllipses]{hello.md}
\end{document}
```

2.3.2 Options

The LaTeX options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` if the =⟨*value*⟩ part has been omitted.

Except for the `plain` option described in Section 2.3.2.1, and the LaTeX themes described in Section 2.3.2.2, and the LaTeX setup snippets described in Section 2.3.2.3, LaTeX options map directly to the options recognized by the plain TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain TeX interface (see Sections 2.2.3 and 2.2.4).

The LaTeX options may be specified when loading the LaTeX package, when using the `markdown*` LaTeX environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/latex-options }
      { #1 }
  }
\let\markdownSetup=\@@_setup:n
\ExplSyntaxOff
```

We may also store LaTeX options as *setup snippets* and invoke them later using the `\markdownSetupSnippet` macro. The `\markdownSetupSnippet` macro receives two arguments: the name of the setup snippet and the options to store:

```
\newcommand\markdownSetupSnippet[2]{%
  \markdownIfSnippetExists{#1}%
    {%
      \markdownWarning
        {Redefined setup snippet \markdownLaTeXThemeName#1}%
      \csname markdownLaTeXSetupSnippet%
        \markdownLaTeXThemeName#1\endcsname={#2}%
    }{%
      \newtoks\next
        \next={#2}%
      \expandafter\let\csname markdownLaTeXSetupSnippet%
        \markdownLaTeXThemeName#1\endcsname=\next
    }}%
```

To decide whether a setup snippet exists, we can use the `\markdownIfSnippetExists` macro:

```
\newcommand\markdownIfSnippetExists[3]{%
  \@ifundefined
```

```
    {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
    {#3}{#2}}%
```

See Section 2.3.2.2 for information on interactions between setup snippets and LaTeX themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of LaTeX options, we will maintain the `\g_@@_latex_options_seq` sequence.

```
\ExplSyntaxOn
\seq_new:N \g_@@_latex_options_seq
```

To enable the reflection of default LaTeX options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```
\prop_new:N \g_@@_latex_option_types_prop
\prop_new:N \g_@@_default_latex_options_prop
\tl_const:Nn \c_@@_option_layer_latex_tl { latex }
\seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
\cs_new:Nn
  \@@_add_latex_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_latex_tl
      { #1 }
      { #2 }
      { #3 }
  }
```

2.3.2.1 No default token renderer prototypes

Default token renderer prototypes require LaTeX packages that may clash with other packages used in a document. Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain TeX implementation (see Section 3.2.2) and prevent the soft LaTeX prerequisites in Section 1.1.3 from being loaded: The plain option must be set before or when loading the package. Setting the option after loading the package will have no effect.

```
\usepackage[plain]{markdown}
```

```
\@@_add_latex_option:nnn
  { plain }
  { boolean }
  { false }
\ExplSyntaxOff
```

2.3.2.2 LaTeX themes

User-defined LaTeX themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to LaTeX packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The LaTeX option `theme`=⟨*theme name*⟩ loads a LaTeX package (further referred to as *a theme*) named `markdowntheme`⟨*munged theme name*⟩`.sty`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (`/`) for an underscore (`_`), the theme *name* is *qualified* and contains no underscores, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer LaTeX package, which provides similar functionality with its `\usetheme` macro [6, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single LaTeX document class or for a single LaTeX package. The preferred format of a theme name is ⟨*theme author*⟩`/`⟨*target LaTeX document class or package*⟩`/`⟨*private naming scheme*⟩, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged, because LaTeX packages are identified only by their filenames, not by their pathnames. [7] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a LaTeX package named `markdownthemewitiko_beamer_MU.sty`.

If the LaTeX option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown LaTeX package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` LaTeX package, and finally the `markdownthemewitiko_dot.sty` LaTeX package:

```
\usepackage[
  theme = witiko/beamer/MU,
  theme = witiko/dot,
]{markdown}
```

```
\newif\ifmarkdownLaTeXLoaded
  \markdownLaTeXLoadedfalse
\AtEndOfPackage{\markdownLaTeXLoadedtrue}
\ExplSyntaxOn
\tl_new:N \markdownLaTeXThemePackageName
```

```
\cs_new:Nn
  \@@_set_latex_theme:n
  {
    \str_if_in:nnF
      { #1 }
      { / }
      {
        \markdownError
        { Won't~load~theme~with~unqualified~name~#1 }
        { Theme~names~must~contain~at~least~one~forward~slash }
      }
    \str_if_in:nnT
      { #1 }
      { _ }
      {
        \markdownError
        { Won't~load~theme~with~an~underscore~in~its~name~#1 }
        { Theme~names~must~not~contain~underscores~in~their~names }
      }
    \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
    \str_replace_all:Nnn
      \markdownLaTeXThemePackageName
      { / }
      { _ }
    \edef\markdownLaTeXThemePackageName{
      markdowntheme\markdownLaTeXThemePackageName}
    \expandafter\markdownLaTeXThemeLoad\expandafter{
      \markdownLaTeXThemePackageName}{#1/}
  }
\keys_define:nn
  { markdown/latex-options }
  {
    theme .code:n = { \@@_set_latex_theme:n { #1 } },
  }
\ExplSyntaxOff
```

The LaTeX themes have a useful synergy with the setup snippets (see Section 2.3.2): To make it less likely that different themes will define setup snippets with the same name, we will prepend ⟨*theme name*⟩`/` before the snippet name and use the result as the snippet name. For example, if the `witiko/dot` theme defines the `product` setup snippet, the setup snippet will be available under the name `witiko/dot/product`.Due to limitations of LaTeX, themes may not be loaded after the beginning of a LaTeX document.

```
\ExplSyntaxOn
\@onlypreamble
  \@@_set_latex_theme:n
\ExplSyntaxOff
```

Example themes provided with the Markdown package include:

witiko/dot A theme that typesets fenced code blocks with the `dot …` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

```
\documentclass{article}
\usepackage[theme=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
  height = 0.65\paperheight,
  keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathemathical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

  latex -> pmml;
  latex -> cmml;
  pmml -> slt;
  cmml -> opt;
  cmml -> prefix;
  cmml -> infix;
  pmml -> mterms [style=dashed];
  cmml -> mterms;

  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
```
\end{markdown}
\end{document}
```

Typesetting the above document produces the output shown in Figure 4.

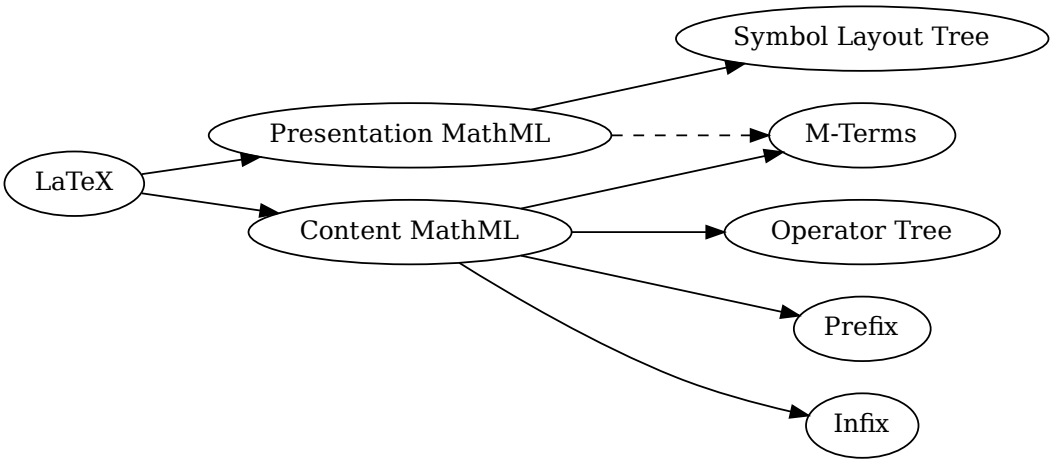

Figure 4: Various formats of mathemathical formulae

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain TEX option is enabled.

```
\ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%
```

witiko/graphicx/http A theme that adds support for downloading images whose URL has the http or https protocol.

```
\documentclass{article}
\usepackage[theme=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}
![img](https://github.com/witiko/markdown/raw/main/markdown.png
       "The banner of the Markdown package")
\end{markdown}
\end{document}
```

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile LATEX package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `\markdownOptionFrozenCache` plain TEX option is enabled.

```
\ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

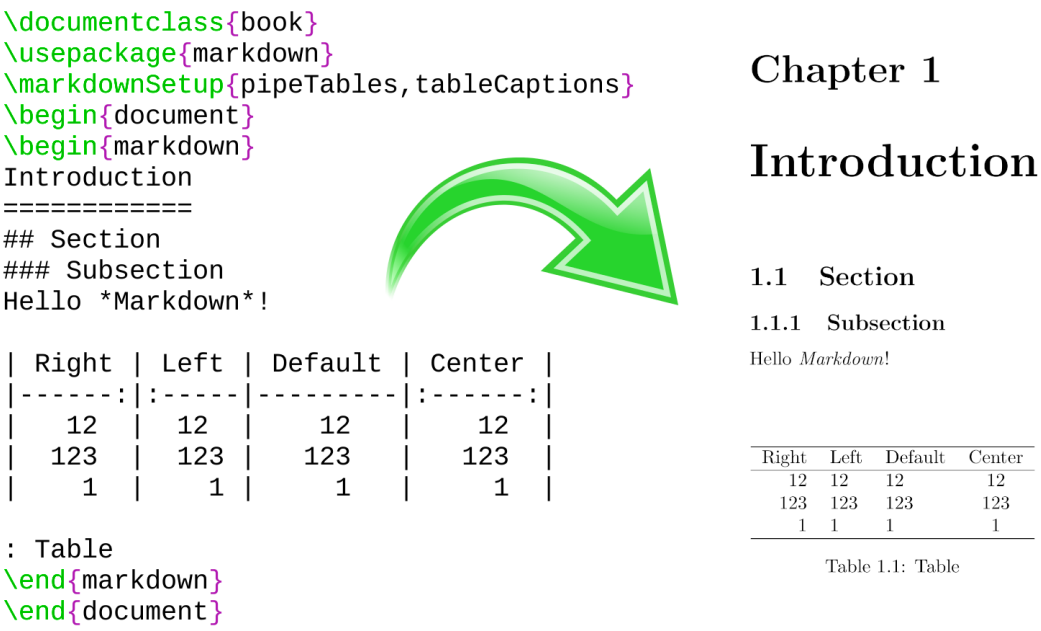

Figure 5: The banner of the Markdown package

witiko/tilde A theme that makes tilde (`~`) always typeset the non-breaking space even when the `hybrid` Lua option is disabled.

```
\documentclass{article}
\usepackage[theme=witiko/tilde]{markdown}
\begin{document}
\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}
```

Typesetting the above document produces the following text: “Bartel Leendert van der Waerden”.

```
\ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%
```

Please, see Section 3.3.2.1 for implementation details of the example themes.

2.3.2.3 LaTeX setup snippets

The LaTeX option with key `snippet` invokes a snippet named ⟨*value*⟩:

```
\ExplSyntaxOn
```

```
\keys_define:nn
  { markdown/latex-options }
  {
    snippet .code:n = {
      \markdownIfSnippetExists{#1}
        {
          \expandafter\markdownSetup\expandafter{
            \the\csname markdownLaTeXSetupSnippet
            \markdownLaTeXThemeName#1\endcsname}
        }{
          \markdownError
            {Can't~invoke~setup~snippet~#1}
            {The~setup~snippet~is~undefined}
        }
    }
  }
\ExplSyntaxOff
```

Here is how we can use setup snippets to store options and invoke them later:

```
\markdownSetupSnippet{romanNumerals}{
  renderers = {
      olItemWithNumber = {\item[\romannumeral#1\relax.]},
  },
}
\begin{markdown}

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

\end{markdown}
\begin{markdown*}{snippet=romanNumerals}

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown*}
```

2.3.2.4 Plain TEX Interface Options

Here, we automatically define plain TEX macros and the ⟨*key*⟩=⟨*value*⟩ interface for the above LATEX options.

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_latex_options_seq
      {
          \@@_plain_tex_define_option_command:n
            { ##1 }
      }
```

Furthermore, we also define the ⟨*key*⟩=⟨*value*⟩ interface for all option macros recognized by the Lua and plain TEX interfaces.

```
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_latex_define_option_keyval:nn
                { ##1 }
                { ####1 }
          }
      }
  }
\cs_new:Nn \@@_latex_define_option_keyval:nn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \keys_define:nn
      { markdown/latex-options }
      {
        #2 .code:n = {
          \@@_set_option_value:nn
            { #2 }
            { ##1 }
        },
      }
    \str_if_eq:VVT
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/latex-options }
          {
            #2 .default:n = { true },
```

```
        }
    }
```

For options of type `clist`, we assume that ⟨*key*⟩ is a regular English noun in plural (such as `extensions`) and we also define the ⟨*singular key*⟩=⟨*value*⟩ interface, where ⟨*singular key*⟩ is ⟨*key*⟩ after stripping the trailing -s (such as `extension`). Rather than setting the option to ⟨*value*⟩, this interface appends ⟨*value*⟩ to the current value as the rightmost item in the list.

```
  \str_if_eq:VVT
    \l_tmpa_tl
    \c_@@_option_type_clist_tl
    {
      \tl_set:Nn
        \l_tmpa_tl
        { #2 }
      \tl_reverse:N
        \l_tmpa_tl
      \str_if_eq:enF
        {
          \tl_head:V
            \l_tmpa_tl
        }
        { s }
        {
            \msg_error:nnn
              { @@ }
              { malformed-name-for-clist-option }
              { #2 }
        }
      \tl_set:Nx
        \l_tmpa_tl
        {
          \tl_tail:V
            \l_tmpa_tl
        }
      \tl_reverse:N
        \l_tmpa_tl
      \tl_put_right:Nn
        \l_tmpa_tl
        {
          .code:n = {
            \@@_get_option_value:nN
              { #2 }
              \l_tmpa_tl
            \clist_set:NV
              \l_tmpa_clist
              { \l_tmpa_tl, { ##1 } }
```

```
              \@@_set_option_value:nV
                { #2 }
                \l_tmpa_clist
            }
          }
        \keys_define:nV
          { markdown/latex-options }
          \l_tmpa_tl
      }
  }
\cs_generate_variant:Nn
  \clist_set:Nn
  { NV }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { F }
\msg_new:nnn
  { @@ }
  { malformed-name-for-clist-option }
  {
    Clist~option~name~#1~does~not~end~with~-s.
  }
\@@_latex_define_option_commands_and_keyvals:
\ExplSyntaxOff
```

The `\markdownOptionFinalizeCache` and `\markdownOptionFrozenCache` plain TEX options are exposed through LATEX options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [8, Section 5.1], which supports the `finalizecache` and `frozencache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document class options. By passing `finalizecache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

```
\documentclass[frozencache]{article}
\usepackage{markdown,minted}
\begin{document}
\end{document}
```

We hope that other packages will support the `finalizecache` and `frozencache` package options in the future, so that they can become a standard interface for preparing LaTeX document sources for distribution.

```
\DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
\DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

The following example LaTeX code showcases a possible configuration of plain TeX interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

```
\markdownSetup{
  hybrid,
  smartEllipses,
  cacheDir = /tmp,
}
```

2.3.2.5 Plain TeX Markdown Token Renderers

The LaTeX interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain TeX interface (see Section 2.2.3).

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_renderers:
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_latex_define_renderer:n
  }
\cs_new:Nn \@@_latex_define_renderer:n
  {
    \@@_renderer_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_latex_define_renderer:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
```

```
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
\cs_new:Nn \@@_latex_define_renderer:nNn
  {
    \keys_define:nn
      { markdown/latex-options/renderers }
      {
        #1 .code:n = {
          \cs_generate_from_arg_count:NNnn
            #2
            \cs_set:Npn
            { #3 }
            { ##1 }
        },
      }
  }
\cs_generate_variant:Nn
  \@@_latex_define_renderer:nNn
  { ncV }
\ExplSyntaxOff
```

The following example LaTeX code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```
\markdownSetup{
  renderers = {
    link = {#4},                    % Render links as the link title.
    emphasis = {\emph{#1}},     % Render emphasized text via `\emph`.
  }
}
```

2.3.2.6 Plain TeX Markdown Token Renderer Prototypes

The LaTeX interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain TeX interface (see Section 2.2.4).

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_renderer_prototypes:
```

```
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_latex_define_renderer_prototype:n
  }
\cs_new:Nn \@@_latex_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_latex_define_renderer_prototype:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_latex_define_renderer_prototype:nNn
  {
    \keys_define:nn
      { markdown/latex-options/renderer-prototypes }
      {
        #1 .code:n = {
          \cs_generate_from_arg_count:NNnn
            #2
            \cs_set:Npn
            { #3 }
            { ##1 }
        },
      }
  }
\cs_generate_variant:Nn
  \@@_latex_define_renderer_prototype:nNn
  { ncV }
\ExplSyntaxOff
```

The following example LaTeX code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```
\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}},    % Render inline code via `\texttt`.
```

```
  }
}
```

2.4 ConTEXt Interface

The ConTEXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTEXt and facilities for setting Lua, plain TEX, and ConTEXt options used during the conversion from markdown to plain TEX. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

```
\writestatus{loading}{ConTeXt User Module / markdown}%
\startmodule[markdown]
\unprotect
```

The ConTEXt implementation redefines the plain TEX logging macros (see Section 3.2.1) to use the ConTEXt `\writestatus` macro.

```
\def\markdownInfo#1{\writestatus{markdown}{#1.}}%
\def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\input markdown/markdown
```

The ConTEXt interface is implemented by the `t-markdown.tex` ConTEXt module file that can be loaded as follows:

```
\usemodule[t][markdown]
```

It is expected that the special plain TEX characters have the expected category codes, when `\input`ting the file.

2.4.1 Typesetting Markdown

The interface exposes the `\startmarkdown` and `\stopmarkdown` macro pair for the typesetting of a markdown document fragment, and defines the `\inputmarkdown` command.

```
\let\startmarkdown\relax
\let\stopmarkdown\relax
\let\inputmarkdown\relax
```

You may prepend your own code to the `\startmarkdown` macro and redefine the `\stopmarkdown` macro to produce special effects before and after the markdown block.

Note that the `\startmarkdown` and `\stopmarkdown` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain TEX interface.

The following example ConTEXt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```
\usemodule[t][markdown]
\starttext
\startmarkdown
_Hello_ **world** ...
\stopmarkdown
\stoptext
```

The `\inputmarkdown` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TEX. Unlike the `\markdownInput` macro provided by the plain TEX interface, this macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influnce this markdown document.

The following example LATEX code showcases the usage of the `\markdownInput` macro:

```
\usemodule[t][markdown]
\starttext
\inputmarkdown[smartEllipses]{hello.md}
\stoptext
```

2.4.2 Options

The ConTEXt options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` (or, equivalently, ⟨*key*⟩=`yes`) if the =⟨*value*⟩ part has been omitted.

ConTEXt options map directly to the options recognized by the plain TEX interface (see Section 2.2.2).

The ConTEXt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/context-options }
      { #1 }
  }
\long\def\setupmarkdown[#1]
  {
    \@@_setup:n
```

```
      { #1 }
  }
\ExplSyntaxOff
```

2.4.2.1 ConTEXt Interface Options

We define the ⟨*key*⟩=⟨*value*⟩ interface for all option macros recognized by the Lua and plain TEX interfaces.

```
\ExplSyntaxOn
\cs_new:Nn \@@_context_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_context_define_option_keyval:nn
                { ##1 }
                { ####1 }
          }
      }
  }
\cs_new:Nn \@@_context_define_option_keyval:nn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \keys_define:nn
      { markdown/context-options }
      {
        #2 .code:n = {
          \tl_set:Nx
            \l_tmpa_tl
            {
              \str_case:nnF
                { ##1 }
                {
                  { yes } { true }
                  { no } { false }
                }
                { ##1 }
            }
          \@@_set_option_value:nV
            { #2 }
            \l_tmpa_tl
        },
```

```
      }
    \str_if_eq:VVT
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/context-options }
          {
            #2 .default:n = { true },
          }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\@@_context_define_option_commands_and_keyvals:
\ExplSyntaxOff
```

3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is performed by the Lua layer. The plain TEX layer provides default definitions for the token renderers. The LATEX and ConTEXt layers correct idiosyncrasies of the respective TEX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain TEX, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain TEX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
local upper, gsub, format, length =
  string.upper, string.gsub, string.format, string.len
local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
  lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
  lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain TEX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
2284 local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

```
2285 function util.err(msg, exit_code)
2286   io.stderr:write("markdown.lua: " .. msg .. "\n")
2287   os.exit(exit_code or 1)
2288 end
```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
2289 function util.cache(dir, string, salt, transform, suffix)
2290   local digest = md5.sumhexa(string .. (salt or ""))
2291   local name = util.pathname(dir, digest .. suffix)
2292   local file = io.open(name, "r")
2293   if file == nil then -- If no cache entry exists, then create a new one.
2294     file = assert(io.open(name, "w"),
2295       [[Could not open file "]] .. name .. [[" for writing]])
2296     local result = string
2297     if transform ~= nil then
2298       result = transform(result)
2299     end
2300     assert(file:write(result))
2301     assert(file:close())
2302   end
2303   return name
2304 end
```

The `util.table_copy` method creates a shallow copy of a table `t` and its metatable.

```
2305 function util.table_copy(t)
2306   local u = { }
2307   for k, v in pairs(t) do u[k] = v end
2308   return setmetatable(u, getmetatable(t))
2309 end
```

The `util.lookup_files` method looks up files with filename `f` and returns its path. If the kpathsea library is available, it will search for files not only in the current working directory but also in the TEX directory structure. Further options for kpathsea can be specified in table `options`. [1, Section 10.7.4]

```
util.lookup_files = (function()
  local ran_ok, kpse = pcall(require, "kpse")
  if ran_ok then
    kpse.set_program_name("luatex")
  else
    kpse = { lookup = function(f, _) return f end }
  end

  local function lookup_files(f, options)
    return kpse.lookup(f, options)
  end

  return lookup_files
end)()
```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [9, Chapter 21].

```
function util.expand_tabs_in_line(s, tabstop)
  local tab = tabstop or 4
  local corr = 0
  return (s:gsub("()\t", function(p)
            local sp = tab - (p - 1 + corr) % tab
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
end
```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```
function util.walk(t, f)
  local typ = type(t)
  if typ == "string" then
    f(t)
  elseif typ == "table" then
    local i = 1
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
```

```
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

The `util.flatten` method flattens an array `ary` that does not contain cycles and returns the result.

```
function util.flatten(ary)
  local new = {}
  for _,v in ipairs(ary) do
    if type(v) == "table" then
      for _,w in ipairs(util.flatten(v)) do
        new[#new + 1] = w
      end
    else
      new[#new + 1] = v
    end
  end
  return new
end
```

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

```
function util.rope_to_string(rope)
  local buffer = {}
  util.walk(rope, function(x) buffer[#buffer + 1] = x end)
  return table.concat(buffer)
end
```

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```
function util.rope_last(rope)
  if #rope == 0 then
    return nil
  else
    local l = rope[#rope]
    if type(l) == "table" then
      return util.rope_last(l)
    else
      return l
    end
  end
end
```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.intersperse(ary, x)
  local new = {}
  local l = #ary
  for i,v in ipairs(ary) do
    local n = #new
    new[n + 1] = v
    if i ~= l then
      new[n + 2] = x
    end
  end
  return new
end
```

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.map(ary, f)
  local new = {}
  for i,v in ipairs(ary) do
    new[i] = f(v)
  end
  return new
end
```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurances of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

```
function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
  local char_escapes_list = ""
  for i,_ in pairs(char_escapes) do
    char_escapes_list = char_escapes_list .. i
  end
```

Create an LPeg capture `escapable` that produces the escaped string corresponding to the matched escapable character.

```
  local escapable = S(char_escapes_list) / char_escapes
```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{\texttt{(k,v)}\in\texttt{string\_escapes}} \texttt{P(k) / v} + \texttt{escapable}$$

capture that replaces any occurance of the string `k` with the string `v` for each $(k, v) \in$ `string_escapes`. Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corrolary, the strings always take precedence over the characters.

```
2410   if string_escapes then
2411     for k,v in pairs(string_escapes) do
2412       escapable = P(k) / v + escapable
2413     end
2414   end
```

Create an LPeg capture `escape_string` that captures anything `escapable` does and matches any other unmatched characters.

```
2415   local escape_string = Cs((escapable + any)^0)
```

Return a function that matches the input string `s` against the `escape_string` capture.

```
2416   return function(s)
2417     return lpeg.match(escape_string, s)
2418   end
2419 end
```

The `util.pathname` method produces a pathname out of a directory name `dir` and a filename `file` and returns it.

```
2420 function util.pathname(dir, file)
2421   if #dir == 0 then
2422     return file
2423   else
2424     return dir .. "/" .. file
2425   end
2426 end
```

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
2427 local entities = {}
2428
2429 local character_entities = {
2430   ["Tab"] = 9,
2431   ["NewLine"] = 10,
2432   ["excl"] = 33,
2433   ["quot"] = 34,
2434   ["QUOT"] = 34,
2435   ["num"] = 35,
2436   ["dollar"] = 36,
2437   ["percnt"] = 37,
```

```
2438  ["amp"] = 38,
2439  ["AMP"] = 38,
2440  ["apos"] = 39,
2441  ["lpar"] = 40,
2442  ["rpar"] = 41,
2443  ["ast"] = 42,
2444  ["midast"] = 42,
2445  ["plus"] = 43,
2446  ["comma"] = 44,
2447  ["period"] = 46,
2448  ["sol"] = 47,
2449  ["colon"] = 58,
2450  ["semi"] = 59,
2451  ["lt"] = 60,
2452  ["LT"] = 60,
2453  ["equals"] = 61,
2454  ["gt"] = 62,
2455  ["GT"] = 62,
2456  ["quest"] = 63,
2457  ["commat"] = 64,
2458  ["lsqb"] = 91,
2459  ["lbrack"] = 91,
2460  ["bsol"] = 92,
2461  ["rsqb"] = 93,
2462  ["rbrack"] = 93,
2463  ["Hat"] = 94,
2464  ["lowbar"] = 95,
2465  ["grave"] = 96,
2466  ["DiacriticalGrave"] = 96,
2467  ["lcub"] = 123,
2468  ["lbrace"] = 123,
2469  ["verbar"] = 124,
2470  ["vert"] = 124,
2471  ["VerticalLine"] = 124,
2472  ["rcub"] = 125,
2473  ["rbrace"] = 125,
2474  ["nbsp"] = 160,
2475  ["NonBreakingSpace"] = 160,
2476  ["iexcl"] = 161,
2477  ["cent"] = 162,
2478  ["pound"] = 163,
2479  ["curren"] = 164,
2480  ["yen"] = 165,
2481  ["brvbar"] = 166,
2482  ["sect"] = 167,
2483  ["Dot"] = 168,
2484  ["die"] = 168,
```

```
2485    ["DoubleDot"] = 168,
2486    ["uml"] = 168,
2487    ["copy"] = 169,
2488    ["COPY"] = 169,
2489    ["ordf"] = 170,
2490    ["laquo"] = 171,
2491    ["not"] = 172,
2492    ["shy"] = 173,
2493    ["reg"] = 174,
2494    ["circledR"] = 174,
2495    ["REG"] = 174,
2496    ["macr"] = 175,
2497    ["OverBar"] = 175,
2498    ["strns"] = 175,
2499    ["deg"] = 176,
2500    ["plusmn"] = 177,
2501    ["pm"] = 177,
2502    ["PlusMinus"] = 177,
2503    ["sup2"] = 178,
2504    ["sup3"] = 179,
2505    ["acute"] = 180,
2506    ["DiacriticalAcute"] = 180,
2507    ["micro"] = 181,
2508    ["para"] = 182,
2509    ["middot"] = 183,
2510    ["centerdot"] = 183,
2511    ["CenterDot"] = 183,
2512    ["cedil"] = 184,
2513    ["Cedilla"] = 184,
2514    ["sup1"] = 185,
2515    ["ordm"] = 186,
2516    ["raquo"] = 187,
2517    ["frac14"] = 188,
2518    ["frac12"] = 189,
2519    ["half"] = 189,
2520    ["frac34"] = 190,
2521    ["iquest"] = 191,
2522    ["Agrave"] = 192,
2523    ["Aacute"] = 193,
2524    ["Acirc"] = 194,
2525    ["Atilde"] = 195,
2526    ["Auml"] = 196,
2527    ["Aring"] = 197,
2528    ["AElig"] = 198,
2529    ["Ccedil"] = 199,
2530    ["Egrave"] = 200,
2531    ["Eacute"] = 201,
```

```
2532    ["Ecirc"] = 202,
2533    ["Euml"] = 203,
2534    ["Igrave"] = 204,
2535    ["Iacute"] = 205,
2536    ["Icirc"] = 206,
2537    ["Iuml"] = 207,
2538    ["ETH"] = 208,
2539    ["Ntilde"] = 209,
2540    ["Ograve"] = 210,
2541    ["Oacute"] = 211,
2542    ["Ocirc"] = 212,
2543    ["Otilde"] = 213,
2544    ["Ouml"] = 214,
2545    ["times"] = 215,
2546    ["Oslash"] = 216,
2547    ["Ugrave"] = 217,
2548    ["Uacute"] = 218,
2549    ["Ucirc"] = 219,
2550    ["Uuml"] = 220,
2551    ["Yacute"] = 221,
2552    ["THORN"] = 222,
2553    ["szlig"] = 223,
2554    ["agrave"] = 224,
2555    ["aacute"] = 225,
2556    ["acirc"] = 226,
2557    ["atilde"] = 227,
2558    ["auml"] = 228,
2559    ["aring"] = 229,
2560    ["aelig"] = 230,
2561    ["ccedil"] = 231,
2562    ["egrave"] = 232,
2563    ["eacute"] = 233,
2564    ["ecirc"] = 234,
2565    ["euml"] = 235,
2566    ["igrave"] = 236,
2567    ["iacute"] = 237,
2568    ["icirc"] = 238,
2569    ["iuml"] = 239,
2570    ["eth"] = 240,
2571    ["ntilde"] = 241,
2572    ["ograve"] = 242,
2573    ["oacute"] = 243,
2574    ["ocirc"] = 244,
2575    ["otilde"] = 245,
2576    ["ouml"] = 246,
2577    ["divide"] = 247,
2578    ["div"] = 247,
```

```
  ["oslash"] = 248,
  ["ugrave"] = 249,
  ["uacute"] = 250,
  ["ucirc"] = 251,
  ["uuml"] = 252,
  ["yacute"] = 253,
  ["thorn"] = 254,
  ["yuml"] = 255,
  ["Amacr"] = 256,
  ["amacr"] = 257,
  ["Abreve"] = 258,
  ["abreve"] = 259,
  ["Aogon"] = 260,
  ["aogon"] = 261,
  ["Cacute"] = 262,
  ["cacute"] = 263,
  ["Ccirc"] = 264,
  ["ccirc"] = 265,
  ["Cdot"] = 266,
  ["cdot"] = 267,
  ["Ccaron"] = 268,
  ["ccaron"] = 269,
  ["Dcaron"] = 270,
  ["dcaron"] = 271,
  ["Dstrok"] = 272,
  ["dstrok"] = 273,
  ["Emacr"] = 274,
  ["emacr"] = 275,
  ["Edot"] = 278,
  ["edot"] = 279,
  ["Eogon"] = 280,
  ["eogon"] = 281,
  ["Ecaron"] = 282,
  ["ecaron"] = 283,
  ["Gcirc"] = 284,
  ["gcirc"] = 285,
  ["Gbreve"] = 286,
  ["gbreve"] = 287,
  ["Gdot"] = 288,
  ["gdot"] = 289,
  ["Gcedil"] = 290,
  ["Hcirc"] = 292,
  ["hcirc"] = 293,
  ["Hstrok"] = 294,
  ["hstrok"] = 295,
  ["Itilde"] = 296,
  ["itilde"] = 297,
```

```
2626   ["Imacr"] = 298,
2627   ["imacr"] = 299,
2628   ["Iogon"] = 302,
2629   ["iogon"] = 303,
2630   ["Idot"] = 304,
2631   ["imath"] = 305,
2632   ["inodot"] = 305,
2633   ["IJlig"] = 306,
2634   ["ijlig"] = 307,
2635   ["Jcirc"] = 308,
2636   ["jcirc"] = 309,
2637   ["Kcedil"] = 310,
2638   ["kcedil"] = 311,
2639   ["kgreen"] = 312,
2640   ["Lacute"] = 313,
2641   ["lacute"] = 314,
2642   ["Lcedil"] = 315,
2643   ["lcedil"] = 316,
2644   ["Lcaron"] = 317,
2645   ["lcaron"] = 318,
2646   ["Lmidot"] = 319,
2647   ["lmidot"] = 320,
2648   ["Lstrok"] = 321,
2649   ["lstrok"] = 322,
2650   ["Nacute"] = 323,
2651   ["nacute"] = 324,
2652   ["Ncedil"] = 325,
2653   ["ncedil"] = 326,
2654   ["Ncaron"] = 327,
2655   ["ncaron"] = 328,
2656   ["napos"] = 329,
2657   ["ENG"] = 330,
2658   ["eng"] = 331,
2659   ["Omacr"] = 332,
2660   ["omacr"] = 333,
2661   ["Odblac"] = 336,
2662   ["odblac"] = 337,
2663   ["OElig"] = 338,
2664   ["oelig"] = 339,
2665   ["Racute"] = 340,
2666   ["racute"] = 341,
2667   ["Rcedil"] = 342,
2668   ["rcedil"] = 343,
2669   ["Rcaron"] = 344,
2670   ["rcaron"] = 345,
2671   ["Sacute"] = 346,
2672   ["sacute"] = 347,
```

```
2673   ["Scirc"] = 348,
2674   ["scirc"] = 349,
2675   ["Scedil"] = 350,
2676   ["scedil"] = 351,
2677   ["Scaron"] = 352,
2678   ["scaron"] = 353,
2679   ["Tcedil"] = 354,
2680   ["tcedil"] = 355,
2681   ["Tcaron"] = 356,
2682   ["tcaron"] = 357,
2683   ["Tstrok"] = 358,
2684   ["tstrok"] = 359,
2685   ["Utilde"] = 360,
2686   ["utilde"] = 361,
2687   ["Umacr"] = 362,
2688   ["umacr"] = 363,
2689   ["Ubreve"] = 364,
2690   ["ubreve"] = 365,
2691   ["Uring"] = 366,
2692   ["uring"] = 367,
2693   ["Udblac"] = 368,
2694   ["udblac"] = 369,
2695   ["Uogon"] = 370,
2696   ["uogon"] = 371,
2697   ["Wcirc"] = 372,
2698   ["wcirc"] = 373,
2699   ["Ycirc"] = 374,
2700   ["ycirc"] = 375,
2701   ["Yuml"] = 376,
2702   ["Zacute"] = 377,
2703   ["zacute"] = 378,
2704   ["Zdot"] = 379,
2705   ["zdot"] = 380,
2706   ["Zcaron"] = 381,
2707   ["zcaron"] = 382,
2708   ["fnof"] = 402,
2709   ["imped"] = 437,
2710   ["gacute"] = 501,
2711   ["jmath"] = 567,
2712   ["circ"] = 710,
2713   ["caron"] = 711,
2714   ["Hacek"] = 711,
2715   ["breve"] = 728,
2716   ["Breve"] = 728,
2717   ["dot"] = 729,
2718   ["DiacriticalDot"] = 729,
2719   ["ring"] = 730,
```

```
2720  ["ogon"] = 731,
2721  ["tilde"] = 732,
2722  ["DiacriticalTilde"] = 732,
2723  ["dblac"] = 733,
2724  ["DiacriticalDoubleAcute"] = 733,
2725  ["DownBreve"] = 785,
2726  ["UnderBar"] = 818,
2727  ["Alpha"] = 913,
2728  ["Beta"] = 914,
2729  ["Gamma"] = 915,
2730  ["Delta"] = 916,
2731  ["Epsilon"] = 917,
2732  ["Zeta"] = 918,
2733  ["Eta"] = 919,
2734  ["Theta"] = 920,
2735  ["Iota"] = 921,
2736  ["Kappa"] = 922,
2737  ["Lambda"] = 923,
2738  ["Mu"] = 924,
2739  ["Nu"] = 925,
2740  ["Xi"] = 926,
2741  ["Omicron"] = 927,
2742  ["Pi"] = 928,
2743  ["Rho"] = 929,
2744  ["Sigma"] = 931,
2745  ["Tau"] = 932,
2746  ["Upsilon"] = 933,
2747  ["Phi"] = 934,
2748  ["Chi"] = 935,
2749  ["Psi"] = 936,
2750  ["Omega"] = 937,
2751  ["alpha"] = 945,
2752  ["beta"] = 946,
2753  ["gamma"] = 947,
2754  ["delta"] = 948,
2755  ["epsiv"] = 949,
2756  ["varepsilon"] = 949,
2757  ["epsilon"] = 949,
2758  ["zeta"] = 950,
2759  ["eta"] = 951,
2760  ["theta"] = 952,
2761  ["iota"] = 953,
2762  ["kappa"] = 954,
2763  ["lambda"] = 955,
2764  ["mu"] = 956,
2765  ["nu"] = 957,
2766  ["xi"] = 958,
```

```
["omicron"] = 959,
["pi"] = 960,
["rho"] = 961,
["sigmav"] = 962,
["varsigma"] = 962,
["sigmaf"] = 962,
["sigma"] = 963,
["tau"] = 964,
["upsi"] = 965,
["upsilon"] = 965,
["phi"] = 966,
["phiv"] = 966,
["varphi"] = 966,
["chi"] = 967,
["psi"] = 968,
["omega"] = 969,
["thetav"] = 977,
["vartheta"] = 977,
["thetasym"] = 977,
["Upsi"] = 978,
["upsih"] = 978,
["straightphi"] = 981,
["piv"] = 982,
["varpi"] = 982,
["Gammad"] = 988,
["gammad"] = 989,
["digamma"] = 989,
["kappav"] = 1008,
["varkappa"] = 1008,
["rhov"] = 1009,
["varrho"] = 1009,
["epsi"] = 1013,
["straightepsilon"] = 1013,
["bepsi"] = 1014,
["backepsilon"] = 1014,
["IOcy"] = 1025,
["DJcy"] = 1026,
["GJcy"] = 1027,
["Jukcy"] = 1028,
["DScy"] = 1029,
["Iukcy"] = 1030,
["YIcy"] = 1031,
["Jsercy"] = 1032,
["LJcy"] = 1033,
["NJcy"] = 1034,
["TSHcy"] = 1035,
["KJcy"] = 1036,
```

```
2814    ["Ubrcy"] = 1038,
2815    ["DZcy"] = 1039,
2816    ["Acy"] = 1040,
2817    ["Bcy"] = 1041,
2818    ["Vcy"] = 1042,
2819    ["Gcy"] = 1043,
2820    ["Dcy"] = 1044,
2821    ["IEcy"] = 1045,
2822    ["ZHcy"] = 1046,
2823    ["Zcy"] = 1047,
2824    ["Icy"] = 1048,
2825    ["Jcy"] = 1049,
2826    ["Kcy"] = 1050,
2827    ["Lcy"] = 1051,
2828    ["Mcy"] = 1052,
2829    ["Ncy"] = 1053,
2830    ["Ocy"] = 1054,
2831    ["Pcy"] = 1055,
2832    ["Rcy"] = 1056,
2833    ["Scy"] = 1057,
2834    ["Tcy"] = 1058,
2835    ["Ucy"] = 1059,
2836    ["Fcy"] = 1060,
2837    ["KHcy"] = 1061,
2838    ["TScy"] = 1062,
2839    ["CHcy"] = 1063,
2840    ["SHcy"] = 1064,
2841    ["SHCHcy"] = 1065,
2842    ["HARDcy"] = 1066,
2843    ["Ycy"] = 1067,
2844    ["SOFTcy"] = 1068,
2845    ["Ecy"] = 1069,
2846    ["YUcy"] = 1070,
2847    ["YAcy"] = 1071,
2848    ["acy"] = 1072,
2849    ["bcy"] = 1073,
2850    ["vcy"] = 1074,
2851    ["gcy"] = 1075,
2852    ["dcy"] = 1076,
2853    ["iecy"] = 1077,
2854    ["zhcy"] = 1078,
2855    ["zcy"] = 1079,
2856    ["icy"] = 1080,
2857    ["jcy"] = 1081,
2858    ["kcy"] = 1082,
2859    ["lcy"] = 1083,
2860    ["mcy"] = 1084,
```

```
["ncy"] = 1085,
["ocy"] = 1086,
["pcy"] = 1087,
["rcy"] = 1088,
["scy"] = 1089,
["tcy"] = 1090,
["ucy"] = 1091,
["fcy"] = 1092,
["khcy"] = 1093,
["tscy"] = 1094,
["chcy"] = 1095,
["shcy"] = 1096,
["shchcy"] = 1097,
["hardcy"] = 1098,
["ycy"] = 1099,
["softcy"] = 1100,
["ecy"] = 1101,
["yucy"] = 1102,
["yacy"] = 1103,
["iocy"] = 1105,
["djcy"] = 1106,
["gjcy"] = 1107,
["jukcy"] = 1108,
["dscy"] = 1109,
["iukcy"] = 1110,
["yicy"] = 1111,
["jsercy"] = 1112,
["ljcy"] = 1113,
["njcy"] = 1114,
["tshcy"] = 1115,
["kjcy"] = 1116,
["ubrcy"] = 1118,
["dzcy"] = 1119,
["ensp"] = 8194,
["emsp"] = 8195,
["emsp13"] = 8196,
["emsp14"] = 8197,
["numsp"] = 8199,
["puncsp"] = 8200,
["thinsp"] = 8201,
["ThinSpace"] = 8201,
["hairsp"] = 8202,
["VeryThinSpace"] = 8202,
["ZeroWidthSpace"] = 8203,
["NegativeVeryThinSpace"] = 8203,
["NegativeThinSpace"] = 8203,
["NegativeMediumSpace"] = 8203,
```

```
2908  ["NegativeThickSpace"] = 8203,
2909  ["zwnj"] = 8204,
2910  ["zwj"] = 8205,
2911  ["lrm"] = 8206,
2912  ["rlm"] = 8207,
2913  ["hyphen"] = 8208,
2914  ["dash"] = 8208,
2915  ["ndash"] = 8211,
2916  ["mdash"] = 8212,
2917  ["horbar"] = 8213,
2918  ["Verbar"] = 8214,
2919  ["Vert"] = 8214,
2920  ["lsquo"] = 8216,
2921  ["OpenCurlyQuote"] = 8216,
2922  ["rsquo"] = 8217,
2923  ["rsquor"] = 8217,
2924  ["CloseCurlyQuote"] = 8217,
2925  ["lsquor"] = 8218,
2926  ["sbquo"] = 8218,
2927  ["ldquo"] = 8220,
2928  ["OpenCurlyDoubleQuote"] = 8220,
2929  ["rdquo"] = 8221,
2930  ["rdquor"] = 8221,
2931  ["CloseCurlyDoubleQuote"] = 8221,
2932  ["ldquor"] = 8222,
2933  ["bdquo"] = 8222,
2934  ["dagger"] = 8224,
2935  ["Dagger"] = 8225,
2936  ["ddagger"] = 8225,
2937  ["bull"] = 8226,
2938  ["bullet"] = 8226,
2939  ["nldr"] = 8229,
2940  ["hellip"] = 8230,
2941  ["mldr"] = 8230,
2942  ["permil"] = 8240,
2943  ["pertenk"] = 8241,
2944  ["prime"] = 8242,
2945  ["Prime"] = 8243,
2946  ["tprime"] = 8244,
2947  ["bprime"] = 8245,
2948  ["backprime"] = 8245,
2949  ["lsaquo"] = 8249,
2950  ["rsaquo"] = 8250,
2951  ["oline"] = 8254,
2952  ["caret"] = 8257,
2953  ["hybull"] = 8259,
2954  ["frasl"] = 8260,
```

```
["bsemi"] = 8271,
["qprime"] = 8279,
["MediumSpace"] = 8287,
["NoBreak"] = 8288,
["ApplyFunction"] = 8289,
["af"] = 8289,
["InvisibleTimes"] = 8290,
["it"] = 8290,
["InvisibleComma"] = 8291,
["ic"] = 8291,
["euro"] = 8364,
["tdot"] = 8411,
["TripleDot"] = 8411,
["DotDot"] = 8412,
["Copf"] = 8450,
["complexes"] = 8450,
["incare"] = 8453,
["gscr"] = 8458,
["hamilt"] = 8459,
["HilbertSpace"] = 8459,
["Hscr"] = 8459,
["Hfr"] = 8460,
["Poincareplane"] = 8460,
["quaternions"] = 8461,
["Hopf"] = 8461,
["planckh"] = 8462,
["planck"] = 8463,
["hbar"] = 8463,
["plankv"] = 8463,
["hslash"] = 8463,
["Iscr"] = 8464,
["imagline"] = 8464,
["image"] = 8465,
["Im"] = 8465,
["imagpart"] = 8465,
["Ifr"] = 8465,
["Lscr"] = 8466,
["lagran"] = 8466,
["Laplacetrf"] = 8466,
["ell"] = 8467,
["Nopf"] = 8469,
["naturals"] = 8469,
["numero"] = 8470,
["copysr"] = 8471,
["weierp"] = 8472,
["wp"] = 8472,
["Popf"] = 8473,
```

```
3002  ["primes"] = 8473,
3003  ["rationals"] = 8474,
3004  ["Qopf"] = 8474,
3005  ["Rscr"] = 8475,
3006  ["realine"] = 8475,
3007  ["real"] = 8476,
3008  ["Re"] = 8476,
3009  ["realpart"] = 8476,
3010  ["Rfr"] = 8476,
3011  ["reals"] = 8477,
3012  ["Ropf"] = 8477,
3013  ["rx"] = 8478,
3014  ["trade"] = 8482,
3015  ["TRADE"] = 8482,
3016  ["integers"] = 8484,
3017  ["Zopf"] = 8484,
3018  ["ohm"] = 8486,
3019  ["mho"] = 8487,
3020  ["Zfr"] = 8488,
3021  ["zeetrf"] = 8488,
3022  ["iiota"] = 8489,
3023  ["angst"] = 8491,
3024  ["bernou"] = 8492,
3025  ["Bernoullis"] = 8492,
3026  ["Bscr"] = 8492,
3027  ["Cfr"] = 8493,
3028  ["Cayleys"] = 8493,
3029  ["escr"] = 8495,
3030  ["Escr"] = 8496,
3031  ["expectation"] = 8496,
3032  ["Fscr"] = 8497,
3033  ["Fouriertrf"] = 8497,
3034  ["phmmat"] = 8499,
3035  ["Mellintrf"] = 8499,
3036  ["Mscr"] = 8499,
3037  ["order"] = 8500,
3038  ["orderof"] = 8500,
3039  ["oscr"] = 8500,
3040  ["alefsym"] = 8501,
3041  ["aleph"] = 8501,
3042  ["beth"] = 8502,
3043  ["gimel"] = 8503,
3044  ["daleth"] = 8504,
3045  ["CapitalDifferentialD"] = 8517,
3046  ["DD"] = 8517,
3047  ["DifferentialD"] = 8518,
3048  ["dd"] = 8518,
```

```
["ExponentialE"] = 8519,
["exponentiale"] = 8519,
["ee"] = 8519,
["ImaginaryI"] = 8520,
["ii"] = 8520,
["frac13"] = 8531,
["frac23"] = 8532,
["frac15"] = 8533,
["frac25"] = 8534,
["frac35"] = 8535,
["frac45"] = 8536,
["frac16"] = 8537,
["frac56"] = 8538,
["frac18"] = 8539,
["frac38"] = 8540,
["frac58"] = 8541,
["frac78"] = 8542,
["larr"] = 8592,
["leftarrow"] = 8592,
["LeftArrow"] = 8592,
["slarr"] = 8592,
["ShortLeftArrow"] = 8592,
["uarr"] = 8593,
["uparrow"] = 8593,
["UpArrow"] = 8593,
["ShortUpArrow"] = 8593,
["rarr"] = 8594,
["rightarrow"] = 8594,
["RightArrow"] = 8594,
["srarr"] = 8594,
["ShortRightArrow"] = 8594,
["darr"] = 8595,
["downarrow"] = 8595,
["DownArrow"] = 8595,
["ShortDownArrow"] = 8595,
["harr"] = 8596,
["leftrightarrow"] = 8596,
["LeftRightArrow"] = 8596,
["varr"] = 8597,
["updownarrow"] = 8597,
["UpDownArrow"] = 8597,
["nwarr"] = 8598,
["UpperLeftArrow"] = 8598,
["nwarrow"] = 8598,
["nearr"] = 8599,
["UpperRightArrow"] = 8599,
["nearrow"] = 8599,
```

```
["searr"] = 8600,
["searrow"] = 8600,
["LowerRightArrow"] = 8600,
["swarr"] = 8601,
["swarrow"] = 8601,
["LowerLeftArrow"] = 8601,
["nlarr"] = 8602,
["nleftarrow"] = 8602,
["nrarr"] = 8603,
["nrightarrow"] = 8603,
["rarrw"] = 8605,
["rightsquigarrow"] = 8605,
["Larr"] = 8606,
["twoheadleftarrow"] = 8606,
["Uarr"] = 8607,
["Rarr"] = 8608,
["twoheadrightarrow"] = 8608,
["Darr"] = 8609,
["larrtl"] = 8610,
["leftarrowtail"] = 8610,
["rarrtl"] = 8611,
["rightarrowtail"] = 8611,
["LeftTeeArrow"] = 8612,
["mapstoleft"] = 8612,
["UpTeeArrow"] = 8613,
["mapstoup"] = 8613,
["map"] = 8614,
["RightTeeArrow"] = 8614,
["mapsto"] = 8614,
["DownTeeArrow"] = 8615,
["mapstodown"] = 8615,
["larrhk"] = 8617,
["hookleftarrow"] = 8617,
["rarrhk"] = 8618,
["hookrightarrow"] = 8618,
["larrlp"] = 8619,
["looparrowleft"] = 8619,
["rarrlp"] = 8620,
["looparrowright"] = 8620,
["harrw"] = 8621,
["leftrightsquigarrow"] = 8621,
["nharr"] = 8622,
["nleftrightarrow"] = 8622,
["lsh"] = 8624,
["Lsh"] = 8624,
["rsh"] = 8625,
["Rsh"] = 8625,
```

```
3143  ["ldsh"] = 8626,
3144  ["rdsh"] = 8627,
3145  ["crarr"] = 8629,
3146  ["cularr"] = 8630,
3147  ["curvearrowleft"] = 8630,
3148  ["curarr"] = 8631,
3149  ["curvearrowright"] = 8631,
3150  ["olarr"] = 8634,
3151  ["circlearrowleft"] = 8634,
3152  ["orarr"] = 8635,
3153  ["circlearrowright"] = 8635,
3154  ["lharu"] = 8636,
3155  ["LeftVector"] = 8636,
3156  ["leftharpoonup"] = 8636,
3157  ["lhard"] = 8637,
3158  ["leftharpoondown"] = 8637,
3159  ["DownLeftVector"] = 8637,
3160  ["uharr"] = 8638,
3161  ["upharpoonright"] = 8638,
3162  ["RightUpVector"] = 8638,
3163  ["uharl"] = 8639,
3164  ["upharpoonleft"] = 8639,
3165  ["LeftUpVector"] = 8639,
3166  ["rharu"] = 8640,
3167  ["RightVector"] = 8640,
3168  ["rightharpoonup"] = 8640,
3169  ["rhard"] = 8641,
3170  ["rightharpoondown"] = 8641,
3171  ["DownRightVector"] = 8641,
3172  ["dharr"] = 8642,
3173  ["RightDownVector"] = 8642,
3174  ["downharpoonright"] = 8642,
3175  ["dharl"] = 8643,
3176  ["LeftDownVector"] = 8643,
3177  ["downharpoonleft"] = 8643,
3178  ["rlarr"] = 8644,
3179  ["rightleftarrows"] = 8644,
3180  ["RightArrowLeftArrow"] = 8644,
3181  ["udarr"] = 8645,
3182  ["UpArrowDownArrow"] = 8645,
3183  ["lrarr"] = 8646,
3184  ["leftrightarrows"] = 8646,
3185  ["LeftArrowRightArrow"] = 8646,
3186  ["llarr"] = 8647,
3187  ["leftleftarrows"] = 8647,
3188  ["uuarr"] = 8648,
3189  ["upuparrows"] = 8648,
```

```
["rrarr"] = 8649,
["rightrightarrows"] = 8649,
["ddarr"] = 8650,
["downdownarrows"] = 8650,
["lrhar"] = 8651,
["ReverseEquilibrium"] = 8651,
["leftrightharpoons"] = 8651,
["rlhar"] = 8652,
["rightleftharpoons"] = 8652,
["Equilibrium"] = 8652,
["nlArr"] = 8653,
["nLeftarrow"] = 8653,
["nhArr"] = 8654,
["nLeftrightarrow"] = 8654,
["nrArr"] = 8655,
["nRightarrow"] = 8655,
["lArr"] = 8656,
["Leftarrow"] = 8656,
["DoubleLeftArrow"] = 8656,
["uArr"] = 8657,
["Uparrow"] = 8657,
["DoubleUpArrow"] = 8657,
["rArr"] = 8658,
["Rightarrow"] = 8658,
["Implies"] = 8658,
["DoubleRightArrow"] = 8658,
["dArr"] = 8659,
["Downarrow"] = 8659,
["DoubleDownArrow"] = 8659,
["hArr"] = 8660,
["Leftrightarrow"] = 8660,
["DoubleLeftRightArrow"] = 8660,
["iff"] = 8660,
["vArr"] = 8661,
["Updownarrow"] = 8661,
["DoubleUpDownArrow"] = 8661,
["nwArr"] = 8662,
["neArr"] = 8663,
["seArr"] = 8664,
["swArr"] = 8665,
["lAarr"] = 8666,
["Lleftarrow"] = 8666,
["rAarr"] = 8667,
["Rrightarrow"] = 8667,
["zigrarr"] = 8669,
["larrb"] = 8676,
["LeftArrowBar"] = 8676,
```

```
["rarrb"] = 8677,
["RightArrowBar"] = 8677,
["duarr"] = 8693,
["DownArrowUpArrow"] = 8693,
["loarr"] = 8701,
["roarr"] = 8702,
["hoarr"] = 8703,
["forall"] = 8704,
["ForAll"] = 8704,
["comp"] = 8705,
["complement"] = 8705,
["part"] = 8706,
["PartialD"] = 8706,
["exist"] = 8707,
["Exists"] = 8707,
["nexist"] = 8708,
["NotExists"] = 8708,
["nexists"] = 8708,
["empty"] = 8709,
["emptyset"] = 8709,
["emptyv"] = 8709,
["varnothing"] = 8709,
["nabla"] = 8711,
["Del"] = 8711,
["isin"] = 8712,
["isinv"] = 8712,
["Element"] = 8712,
["in"] = 8712,
["notin"] = 8713,
["NotElement"] = 8713,
["notinva"] = 8713,
["niv"] = 8715,
["ReverseElement"] = 8715,
["ni"] = 8715,
["SuchThat"] = 8715,
["notni"] = 8716,
["notniva"] = 8716,
["NotReverseElement"] = 8716,
["prod"] = 8719,
["Product"] = 8719,
["coprod"] = 8720,
["Coproduct"] = 8720,
["sum"] = 8721,
["Sum"] = 8721,
["minus"] = 8722,
["mnplus"] = 8723,
["mp"] = 8723,
```

```
["MinusPlus"] = 8723,
["plusdo"] = 8724,
["dotplus"] = 8724,
["setmn"] = 8726,
["setminus"] = 8726,
["Backslash"] = 8726,
["ssetmn"] = 8726,
["smallsetminus"] = 8726,
["lowast"] = 8727,
["compfn"] = 8728,
["SmallCircle"] = 8728,
["radic"] = 8730,
["Sqrt"] = 8730,
["prop"] = 8733,
["propto"] = 8733,
["Proportional"] = 8733,
["vprop"] = 8733,
["varpropto"] = 8733,
["infin"] = 8734,
["angrt"] = 8735,
["ang"] = 8736,
["angle"] = 8736,
["angmsd"] = 8737,
["measuredangle"] = 8737,
["angsph"] = 8738,
["mid"] = 8739,
["VerticalBar"] = 8739,
["smid"] = 8739,
["shortmid"] = 8739,
["nmid"] = 8740,
["NotVerticalBar"] = 8740,
["nsmid"] = 8740,
["nshortmid"] = 8740,
["par"] = 8741,
["parallel"] = 8741,
["DoubleVerticalBar"] = 8741,
["spar"] = 8741,
["shortparallel"] = 8741,
["npar"] = 8742,
["nparallel"] = 8742,
["NotDoubleVerticalBar"] = 8742,
["nspar"] = 8742,
["nshortparallel"] = 8742,
["and"] = 8743,
["wedge"] = 8743,
["or"] = 8744,
["vee"] = 8744,
```

```
["cap"] = 8745,
["cup"] = 8746,
["int"] = 8747,
["Integral"] = 8747,
["Int"] = 8748,
["tint"] = 8749,
["iiint"] = 8749,
["conint"] = 8750,
["oint"] = 8750,
["ContourIntegral"] = 8750,
["Conint"] = 8751,
["DoubleContourIntegral"] = 8751,
["Cconint"] = 8752,
["cwint"] = 8753,
["cwconint"] = 8754,
["ClockwiseContourIntegral"] = 8754,
["awconint"] = 8755,
["CounterClockwiseContourIntegral"] = 8755,
["there4"] = 8756,
["therefore"] = 8756,
["Therefore"] = 8756,
["becaus"] = 8757,
["because"] = 8757,
["Because"] = 8757,
["ratio"] = 8758,
["Colon"] = 8759,
["Proportion"] = 8759,
["minusd"] = 8760,
["dotminus"] = 8760,
["mDDot"] = 8762,
["homtht"] = 8763,
["sim"] = 8764,
["Tilde"] = 8764,
["thksim"] = 8764,
["thicksim"] = 8764,
["bsim"] = 8765,
["backsim"] = 8765,
["ac"] = 8766,
["mstpos"] = 8766,
["acd"] = 8767,
["wreath"] = 8768,
["VerticalTilde"] = 8768,
["wr"] = 8768,
["nsim"] = 8769,
["NotTilde"] = 8769,
["esim"] = 8770,
["EqualTilde"] = 8770,
```

```
["eqsim"] = 8770,
["sime"] = 8771,
["TildeEqual"] = 8771,
["simeq"] = 8771,
["nsime"] = 8772,
["nsimeq"] = 8772,
["NotTildeEqual"] = 8772,
["cong"] = 8773,
["TildeFullEqual"] = 8773,
["simne"] = 8774,
["ncong"] = 8775,
["NotTildeFullEqual"] = 8775,
["asymp"] = 8776,
["ap"] = 8776,
["TildeTilde"] = 8776,
["approx"] = 8776,
["thkap"] = 8776,
["thickapprox"] = 8776,
["nap"] = 8777,
["NotTildeTilde"] = 8777,
["napprox"] = 8777,
["ape"] = 8778,
["approxeq"] = 8778,
["apid"] = 8779,
["bcong"] = 8780,
["backcong"] = 8780,
["asympeq"] = 8781,
["CupCap"] = 8781,
["bump"] = 8782,
["HumpDownHump"] = 8782,
["Bumpeq"] = 8782,
["bumpe"] = 8783,
["HumpEqual"] = 8783,
["bumpeq"] = 8783,
["esdot"] = 8784,
["DotEqual"] = 8784,
["doteq"] = 8784,
["eDot"] = 8785,
["doteqdot"] = 8785,
["efDot"] = 8786,
["fallingdotseq"] = 8786,
["erDot"] = 8787,
["risingdotseq"] = 8787,
["colone"] = 8788,
["coloneq"] = 8788,
["Assign"] = 8788,
["ecolon"] = 8789,
```

```
["eqcolon"] = 8789,
["ecir"] = 8790,
["eqcirc"] = 8790,
["cire"] = 8791,
["circeq"] = 8791,
["wedgeq"] = 8793,
["veeeq"] = 8794,
["trie"] = 8796,
["triangleq"] = 8796,
["equest"] = 8799,
["questeq"] = 8799,
["ne"] = 8800,
["NotEqual"] = 8800,
["equiv"] = 8801,
["Congruent"] = 8801,
["nequiv"] = 8802,
["NotCongruent"] = 8802,
["le"] = 8804,
["leq"] = 8804,
["ge"] = 8805,
["GreaterEqual"] = 8805,
["geq"] = 8805,
["lE"] = 8806,
["LessFullEqual"] = 8806,
["leqq"] = 8806,
["gE"] = 8807,
["GreaterFullEqual"] = 8807,
["geqq"] = 8807,
["lnE"] = 8808,
["lneqq"] = 8808,
["gnE"] = 8809,
["gneqq"] = 8809,
["Lt"] = 8810,
["NestedLessLess"] = 8810,
["ll"] = 8810,
["Gt"] = 8811,
["NestedGreaterGreater"] = 8811,
["gg"] = 8811,
["twixt"] = 8812,
["between"] = 8812,
["NotCupCap"] = 8813,
["nlt"] = 8814,
["NotLess"] = 8814,
["nless"] = 8814,
["ngt"] = 8815,
["NotGreater"] = 8815,
["ngtr"] = 8815,
```

```
["nle"] = 8816,
["NotLessEqual"] = 8816,
["nleq"] = 8816,
["nge"] = 8817,
["NotGreaterEqual"] = 8817,
["ngeq"] = 8817,
["lsim"] = 8818,
["LessTilde"] = 8818,
["lesssim"] = 8818,
["gsim"] = 8819,
["gtrsim"] = 8819,
["GreaterTilde"] = 8819,
["nlsim"] = 8820,
["NotLessTilde"] = 8820,
["ngsim"] = 8821,
["NotGreaterTilde"] = 8821,
["lg"] = 8822,
["lessgtr"] = 8822,
["LessGreater"] = 8822,
["gl"] = 8823,
["gtrless"] = 8823,
["GreaterLess"] = 8823,
["ntlg"] = 8824,
["NotLessGreater"] = 8824,
["ntgl"] = 8825,
["NotGreaterLess"] = 8825,
["pr"] = 8826,
["Precedes"] = 8826,
["prec"] = 8826,
["sc"] = 8827,
["Succeeds"] = 8827,
["succ"] = 8827,
["prcue"] = 8828,
["PrecedesSlantEqual"] = 8828,
["preccurlyeq"] = 8828,
["sccue"] = 8829,
["SucceedsSlantEqual"] = 8829,
["succcurlyeq"] = 8829,
["prsim"] = 8830,
["precsim"] = 8830,
["PrecedesTilde"] = 8830,
["scsim"] = 8831,
["succsim"] = 8831,
["SucceedsTilde"] = 8831,
["npr"] = 8832,
["nprec"] = 8832,
["NotPrecedes"] = 8832,
```

```
["nsc"] = 8833,
["nsucc"] = 8833,
["NotSucceeds"] = 8833,
["sub"] = 8834,
["subset"] = 8834,
["sup"] = 8835,
["supset"] = 8835,
["Superset"] = 8835,
["nsub"] = 8836,
["nsup"] = 8837,
["sube"] = 8838,
["SubsetEqual"] = 8838,
["subseteq"] = 8838,
["supe"] = 8839,
["supseteq"] = 8839,
["SupersetEqual"] = 8839,
["nsube"] = 8840,
["nsubseteq"] = 8840,
["NotSubsetEqual"] = 8840,
["nsupe"] = 8841,
["nsupseteq"] = 8841,
["NotSupersetEqual"] = 8841,
["subne"] = 8842,
["subsetneq"] = 8842,
["supne"] = 8843,
["supsetneq"] = 8843,
["cupdot"] = 8845,
["uplus"] = 8846,
["UnionPlus"] = 8846,
["sqsub"] = 8847,
["SquareSubset"] = 8847,
["sqsubset"] = 8847,
["sqsup"] = 8848,
["SquareSuperset"] = 8848,
["sqsupset"] = 8848,
["sqsube"] = 8849,
["SquareSubsetEqual"] = 8849,
["sqsubseteq"] = 8849,
["sqsupe"] = 8850,
["SquareSupersetEqual"] = 8850,
["sqsupseteq"] = 8850,
["sqcap"] = 8851,
["SquareIntersection"] = 8851,
["sqcup"] = 8852,
["SquareUnion"] = 8852,
["oplus"] = 8853,
["CirclePlus"] = 8853,
```

```
["ominus"] = 8854,
["CircleMinus"] = 8854,
["otimes"] = 8855,
["CircleTimes"] = 8855,
["osol"] = 8856,
["odot"] = 8857,
["CircleDot"] = 8857,
["ocir"] = 8858,
["circledcirc"] = 8858,
["oast"] = 8859,
["circledast"] = 8859,
["odash"] = 8861,
["circleddash"] = 8861,
["plusb"] = 8862,
["boxplus"] = 8862,
["minusb"] = 8863,
["boxminus"] = 8863,
["timesb"] = 8864,
["boxtimes"] = 8864,
["sdotb"] = 8865,
["dotsquare"] = 8865,
["vdash"] = 8866,
["RightTee"] = 8866,
["dashv"] = 8867,
["LeftTee"] = 8867,
["top"] = 8868,
["DownTee"] = 8868,
["bottom"] = 8869,
["bot"] = 8869,
["perp"] = 8869,
["UpTee"] = 8869,
["models"] = 8871,
["vDash"] = 8872,
["DoubleRightTee"] = 8872,
["Vdash"] = 8873,
["Vvdash"] = 8874,
["VDash"] = 8875,
["nvdash"] = 8876,
["nvDash"] = 8877,
["nVdash"] = 8878,
["nVDash"] = 8879,
["prurel"] = 8880,
["vltri"] = 8882,
["vartriangleleft"] = 8882,
["LeftTriangle"] = 8882,
["vrtri"] = 8883,
["vartriangleright"] = 8883,
```

```
["RightTriangle"] = 8883,
["ltrie"] = 8884,
["trianglelefteq"] = 8884,
["LeftTriangleEqual"] = 8884,
["rtrie"] = 8885,
["trianglerighteq"] = 8885,
["RightTriangleEqual"] = 8885,
["origof"] = 8886,
["imof"] = 8887,
["mumap"] = 8888,
["multimap"] = 8888,
["hercon"] = 8889,
["intcal"] = 8890,
["intercal"] = 8890,
["veebar"] = 8891,
["barvee"] = 8893,
["angrtvb"] = 8894,
["lrtri"] = 8895,
["xwedge"] = 8896,
["Wedge"] = 8896,
["bigwedge"] = 8896,
["xvee"] = 8897,
["Vee"] = 8897,
["bigvee"] = 8897,
["xcap"] = 8898,
["Intersection"] = 8898,
["bigcap"] = 8898,
["xcup"] = 8899,
["Union"] = 8899,
["bigcup"] = 8899,
["diam"] = 8900,
["diamond"] = 8900,
["Diamond"] = 8900,
["sdot"] = 8901,
["sstarf"] = 8902,
["Star"] = 8902,
["divonx"] = 8903,
["divideontimes"] = 8903,
["bowtie"] = 8904,
["ltimes"] = 8905,
["rtimes"] = 8906,
["lthree"] = 8907,
["leftthreetimes"] = 8907,
["rthree"] = 8908,
["rightthreetimes"] = 8908,
["bsime"] = 8909,
["backsimeq"] = 8909,
```

```
["cuvee"] = 8910,
["curlyvee"] = 8910,
["cuwed"] = 8911,
["curlywedge"] = 8911,
["Sub"] = 8912,
["Subset"] = 8912,
["Sup"] = 8913,
["Supset"] = 8913,
["Cap"] = 8914,
["Cup"] = 8915,
["fork"] = 8916,
["pitchfork"] = 8916,
["epar"] = 8917,
["ltdot"] = 8918,
["lessdot"] = 8918,
["gtdot"] = 8919,
["gtrdot"] = 8919,
["Ll"] = 8920,
["Gg"] = 8921,
["ggg"] = 8921,
["leg"] = 8922,
["LessEqualGreater"] = 8922,
["lesseqgtr"] = 8922,
["gel"] = 8923,
["gtreqless"] = 8923,
["GreaterEqualLess"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["nprcue"] = 8928,
["NotPrecedesSlantEqual"] = 8928,
["nsccue"] = 8929,
["NotSucceedsSlantEqual"] = 8929,
["nsqsube"] = 8930,
["NotSquareSubsetEqual"] = 8930,
["nsqsupe"] = 8931,
["NotSquareSupersetEqual"] = 8931,
["lnsim"] = 8934,
["gnsim"] = 8935,
["prnsim"] = 8936,
["precnsim"] = 8936,
["scnsim"] = 8937,
["succnsim"] = 8937,
["nltri"] = 8938,
["ntriangleleft"] = 8938,
["NotLeftTriangle"] = 8938,
```

```
3707  ["nrtri"] = 8939,
3708  ["ntriangleright"] = 8939,
3709  ["NotRightTriangle"] = 8939,
3710  ["nltrie"] = 8940,
3711  ["ntrianglelefteq"] = 8940,
3712  ["NotLeftTriangleEqual"] = 8940,
3713  ["nrtrie"] = 8941,
3714  ["ntrianglerighteq"] = 8941,
3715  ["NotRightTriangleEqual"] = 8941,
3716  ["vellip"] = 8942,
3717  ["ctdot"] = 8943,
3718  ["utdot"] = 8944,
3719  ["dtdot"] = 8945,
3720  ["disin"] = 8946,
3721  ["isinsv"] = 8947,
3722  ["isins"] = 8948,
3723  ["isindot"] = 8949,
3724  ["notinvc"] = 8950,
3725  ["notinvb"] = 8951,
3726  ["isinE"] = 8953,
3727  ["nisd"] = 8954,
3728  ["xnis"] = 8955,
3729  ["nis"] = 8956,
3730  ["notnivc"] = 8957,
3731  ["notnivb"] = 8958,
3732  ["barwed"] = 8965,
3733  ["barwedge"] = 8965,
3734  ["Barwed"] = 8966,
3735  ["doublebarwedge"] = 8966,
3736  ["lceil"] = 8968,
3737  ["LeftCeiling"] = 8968,
3738  ["rceil"] = 8969,
3739  ["RightCeiling"] = 8969,
3740  ["lfloor"] = 8970,
3741  ["LeftFloor"] = 8970,
3742  ["rfloor"] = 8971,
3743  ["RightFloor"] = 8971,
3744  ["drcrop"] = 8972,
3745  ["dlcrop"] = 8973,
3746  ["urcrop"] = 8974,
3747  ["ulcrop"] = 8975,
3748  ["bnot"] = 8976,
3749  ["profline"] = 8978,
3750  ["profsurf"] = 8979,
3751  ["telrec"] = 8981,
3752  ["target"] = 8982,
3753  ["ulcorn"] = 8988,
```

```
  ["ulcorner"] = 8988,
  ["urcorn"] = 8989,
  ["urcorner"] = 8989,
  ["dlcorn"] = 8990,
  ["llcorner"] = 8990,
  ["drcorn"] = 8991,
  ["lrcorner"] = 8991,
  ["frown"] = 8994,
  ["sfrown"] = 8994,
  ["smile"] = 8995,
  ["ssmile"] = 8995,
  ["cylcty"] = 9005,
  ["profalar"] = 9006,
  ["topbot"] = 9014,
  ["ovbar"] = 9021,
  ["solbar"] = 9023,
  ["angzarr"] = 9084,
  ["lmoust"] = 9136,
  ["lmoustache"] = 9136,
  ["rmoust"] = 9137,
  ["rmoustache"] = 9137,
  ["tbrk"] = 9140,
  ["OverBracket"] = 9140,
  ["bbrk"] = 9141,
  ["UnderBracket"] = 9141,
  ["bbrktbrk"] = 9142,
  ["OverParenthesis"] = 9180,
  ["UnderParenthesis"] = 9181,
  ["OverBrace"] = 9182,
  ["UnderBrace"] = 9183,
  ["trpezium"] = 9186,
  ["elinters"] = 9191,
  ["blank"] = 9251,
  ["oS"] = 9416,
  ["circledS"] = 9416,
  ["boxh"] = 9472,
  ["HorizontalLine"] = 9472,
  ["boxv"] = 9474,
  ["boxdr"] = 9484,
  ["boxdl"] = 9488,
  ["boxur"] = 9492,
  ["boxul"] = 9496,
  ["boxvr"] = 9500,
  ["boxvl"] = 9508,
  ["boxhd"] = 9516,
  ["boxhu"] = 9524,
  ["boxvh"] = 9532,
```

```
  ["boxH"] = 9552,
  ["boxV"] = 9553,
  ["boxdR"] = 9554,
  ["boxDr"] = 9555,
  ["boxDR"] = 9556,
  ["boxdL"] = 9557,
  ["boxDl"] = 9558,
  ["boxDL"] = 9559,
  ["boxuR"] = 9560,
  ["boxUr"] = 9561,
  ["boxUR"] = 9562,
  ["boxuL"] = 9563,
  ["boxUl"] = 9564,
  ["boxUL"] = 9565,
  ["boxvR"] = 9566,
  ["boxVr"] = 9567,
  ["boxVR"] = 9568,
  ["boxvL"] = 9569,
  ["boxVl"] = 9570,
  ["boxVL"] = 9571,
  ["boxHd"] = 9572,
  ["boxhD"] = 9573,
  ["boxHD"] = 9574,
  ["boxHu"] = 9575,
  ["boxhU"] = 9576,
  ["boxHU"] = 9577,
  ["boxvH"] = 9578,
  ["boxVh"] = 9579,
  ["boxVH"] = 9580,
  ["uhblk"] = 9600,
  ["lhblk"] = 9604,
  ["block"] = 9608,
  ["blk14"] = 9617,
  ["blk12"] = 9618,
  ["blk34"] = 9619,
  ["squ"] = 9633,
  ["square"] = 9633,
  ["Square"] = 9633,
  ["squf"] = 9642,
  ["squarf"] = 9642,
  ["blacksquare"] = 9642,
  ["FilledVerySmallSquare"] = 9642,
  ["EmptyVerySmallSquare"] = 9643,
  ["rect"] = 9645,
  ["marker"] = 9646,
  ["fltns"] = 9649,
  ["xutri"] = 9651,
```

```
["bigtriangleup"] = 9651,
["utrif"] = 9652,
["blacktriangle"] = 9652,
["utri"] = 9653,
["triangle"] = 9653,
["rtrif"] = 9656,
["blacktriangleright"] = 9656,
["rtri"] = 9657,
["triangleright"] = 9657,
["xdtri"] = 9661,
["bigtriangledown"] = 9661,
["dtrif"] = 9662,
["blacktriangledown"] = 9662,
["dtri"] = 9663,
["triangledown"] = 9663,
["ltrif"] = 9666,
["blacktriangleleft"] = 9666,
["ltri"] = 9667,
["triangleleft"] = 9667,
["loz"] = 9674,
["lozenge"] = 9674,
["cir"] = 9675,
["tridot"] = 9708,
["xcirc"] = 9711,
["bigcirc"] = 9711,
["ultri"] = 9720,
["urtri"] = 9721,
["lltri"] = 9722,
["EmptySmallSquare"] = 9723,
["FilledSmallSquare"] = 9724,
["starf"] = 9733,
["bigstar"] = 9733,
["star"] = 9734,
["phone"] = 9742,
["female"] = 9792,
["male"] = 9794,
["spades"] = 9824,
["spadesuit"] = 9824,
["clubs"] = 9827,
["clubsuit"] = 9827,
["hearts"] = 9829,
["heartsuit"] = 9829,
["diams"] = 9830,
["diamondsuit"] = 9830,
["sung"] = 9834,
["flat"] = 9837,
["natur"] = 9838,
```

```
["natural"] = 9838,
["sharp"] = 9839,
["check"] = 10003,
["checkmark"] = 10003,
["cross"] = 10007,
["malt"] = 10016,
["maltese"] = 10016,
["sext"] = 10038,
["VerticalSeparator"] = 10072,
["lbbrk"] = 10098,
["rbbrk"] = 10099,
["lobrk"] = 10214,
["LeftDoubleBracket"] = 10214,
["robrk"] = 10215,
["RightDoubleBracket"] = 10215,
["lang"] = 10216,
["LeftAngleBracket"] = 10216,
["langle"] = 10216,
["rang"] = 10217,
["RightAngleBracket"] = 10217,
["rangle"] = 10217,
["Lang"] = 10218,
["Rang"] = 10219,
["loang"] = 10220,
["roang"] = 10221,
["xlarr"] = 10229,
["longleftarrow"] = 10229,
["LongLeftArrow"] = 10229,
["xrarr"] = 10230,
["longrightarrow"] = 10230,
["LongRightArrow"] = 10230,
["xharr"] = 10231,
["longleftrightarrow"] = 10231,
["LongLeftRightArrow"] = 10231,
["xlArr"] = 10232,
["Longleftarrow"] = 10232,
["DoubleLongLeftArrow"] = 10232,
["xrArr"] = 10233,
["Longrightarrow"] = 10233,
["DoubleLongRightArrow"] = 10233,
["xhArr"] = 10234,
["Longleftrightarrow"] = 10234,
["DoubleLongLeftRightArrow"] = 10234,
["xmap"] = 10236,
["longmapsto"] = 10236,
["dzigrarr"] = 10239,
["nvlArr"] = 10498,
```

```
["nvrArr"] = 10499,
["nvHarr"] = 10500,
["Map"] = 10501,
["lbarr"] = 10508,
["rbarr"] = 10509,
["bkarow"] = 10509,
["lBarr"] = 10510,
["rBarr"] = 10511,
["dbkarow"] = 10511,
["RBarr"] = 10512,
["drbkarow"] = 10512,
["DDotrahd"] = 10513,
["UpArrowBar"] = 10514,
["DownArrowBar"] = 10515,
["Rarrtl"] = 10518,
["latail"] = 10521,
["ratail"] = 10522,
["lAtail"] = 10523,
["rAtail"] = 10524,
["larrfs"] = 10525,
["rarrfs"] = 10526,
["larrbfs"] = 10527,
["rarrbfs"] = 10528,
["nwarhk"] = 10531,
["nearhk"] = 10532,
["searhk"] = 10533,
["hksearow"] = 10533,
["swarhk"] = 10534,
["hkswarow"] = 10534,
["nwnear"] = 10535,
["nesear"] = 10536,
["toea"] = 10536,
["seswar"] = 10537,
["tosa"] = 10537,
["swnwar"] = 10538,
["rarrc"] = 10547,
["cudarrr"] = 10549,
["ldca"] = 10550,
["rdca"] = 10551,
["cudarrl"] = 10552,
["larrpl"] = 10553,
["curarrm"] = 10556,
["cularrp"] = 10557,
["rarrpl"] = 10565,
["harrcir"] = 10568,
["Uarrocir"] = 10569,
["lurdshar"] = 10570,
```

```
["ldrushar"] = 10571,
["LeftRightVector"] = 10574,
["RightUpDownVector"] = 10575,
["DownLeftRightVector"] = 10576,
["LeftUpDownVector"] = 10577,
["LeftVectorBar"] = 10578,
["RightVectorBar"] = 10579,
["RightUpVectorBar"] = 10580,
["RightDownVectorBar"] = 10581,
["DownLeftVectorBar"] = 10582,
["DownRightVectorBar"] = 10583,
["LeftUpVectorBar"] = 10584,
["LeftDownVectorBar"] = 10585,
["LeftTeeVector"] = 10586,
["RightTeeVector"] = 10587,
["RightUpTeeVector"] = 10588,
["RightDownTeeVector"] = 10589,
["DownLeftTeeVector"] = 10590,
["DownRightTeeVector"] = 10591,
["LeftUpTeeVector"] = 10592,
["LeftDownTeeVector"] = 10593,
["lHar"] = 10594,
["uHar"] = 10595,
["rHar"] = 10596,
["dHar"] = 10597,
["luruhar"] = 10598,
["ldrdhar"] = 10599,
["ruluhar"] = 10600,
["rdldhar"] = 10601,
["lharul"] = 10602,
["llhard"] = 10603,
["rharul"] = 10604,
["lrhard"] = 10605,
["udhar"] = 10606,
["UpEquilibrium"] = 10606,
["duhar"] = 10607,
["ReverseUpEquilibrium"] = 10607,
["RoundImplies"] = 10608,
["erarr"] = 10609,
["simrarr"] = 10610,
["larrsim"] = 10611,
["rarrsim"] = 10612,
["rarrap"] = 10613,
["ltlarr"] = 10614,
["gtrarr"] = 10616,
["subrarr"] = 10617,
["suplarr"] = 10619,
```

```
4036  ["lfisht"] = 10620,
4037  ["rfisht"] = 10621,
4038  ["ufisht"] = 10622,
4039  ["dfisht"] = 10623,
4040  ["lopar"] = 10629,
4041  ["ropar"] = 10630,
4042  ["lbrke"] = 10635,
4043  ["rbrke"] = 10636,
4044  ["lbrkslu"] = 10637,
4045  ["rbrksld"] = 10638,
4046  ["lbrksld"] = 10639,
4047  ["rbrkslu"] = 10640,
4048  ["langd"] = 10641,
4049  ["rangd"] = 10642,
4050  ["lparlt"] = 10643,
4051  ["rpargt"] = 10644,
4052  ["gtlPar"] = 10645,
4053  ["ltrPar"] = 10646,
4054  ["vzigzag"] = 10650,
4055  ["vangrt"] = 10652,
4056  ["angrtvbd"] = 10653,
4057  ["ange"] = 10660,
4058  ["range"] = 10661,
4059  ["dwangle"] = 10662,
4060  ["uwangle"] = 10663,
4061  ["angmsdaa"] = 10664,
4062  ["angmsdab"] = 10665,
4063  ["angmsdac"] = 10666,
4064  ["angmsdad"] = 10667,
4065  ["angmsdae"] = 10668,
4066  ["angmsdaf"] = 10669,
4067  ["angmsdag"] = 10670,
4068  ["angmsdah"] = 10671,
4069  ["bemptyv"] = 10672,
4070  ["demptyv"] = 10673,
4071  ["cemptyv"] = 10674,
4072  ["raemptyv"] = 10675,
4073  ["laemptyv"] = 10676,
4074  ["ohbar"] = 10677,
4075  ["omid"] = 10678,
4076  ["opar"] = 10679,
4077  ["operp"] = 10681,
4078  ["olcross"] = 10683,
4079  ["odsold"] = 10684,
4080  ["olcir"] = 10686,
4081  ["ofcir"] = 10687,
4082  ["olt"] = 10688,
```

```
4083  ["ogt"] = 10689,
4084  ["cirscir"] = 10690,
4085  ["cirE"] = 10691,
4086  ["solb"] = 10692,
4087  ["bsolb"] = 10693,
4088  ["boxbox"] = 10697,
4089  ["trisb"] = 10701,
4090  ["rtriltri"] = 10702,
4091  ["LeftTriangleBar"] = 10703,
4092  ["RightTriangleBar"] = 10704,
4093  ["race"] = 10714,
4094  ["iinfin"] = 10716,
4095  ["infintie"] = 10717,
4096  ["nvinfin"] = 10718,
4097  ["eparsl"] = 10723,
4098  ["smeparsl"] = 10724,
4099  ["eqvparsl"] = 10725,
4100  ["lozf"] = 10731,
4101  ["blacklozenge"] = 10731,
4102  ["RuleDelayed"] = 10740,
4103  ["dsol"] = 10742,
4104  ["xodot"] = 10752,
4105  ["bigodot"] = 10752,
4106  ["xoplus"] = 10753,
4107  ["bigoplus"] = 10753,
4108  ["xotime"] = 10754,
4109  ["bigotimes"] = 10754,
4110  ["xuplus"] = 10756,
4111  ["biguplus"] = 10756,
4112  ["xsqcup"] = 10758,
4113  ["bigsqcup"] = 10758,
4114  ["qint"] = 10764,
4115  ["iiiint"] = 10764,
4116  ["fpartint"] = 10765,
4117  ["cirfnint"] = 10768,
4118  ["awint"] = 10769,
4119  ["rppolint"] = 10770,
4120  ["scpolint"] = 10771,
4121  ["npolint"] = 10772,
4122  ["pointint"] = 10773,
4123  ["quatint"] = 10774,
4124  ["intlarhk"] = 10775,
4125  ["pluscir"] = 10786,
4126  ["plusacir"] = 10787,
4127  ["simplus"] = 10788,
4128  ["plusdu"] = 10789,
4129  ["plussim"] = 10790,
```

```
  ["plustwo"] = 10791,
  ["mcomma"] = 10793,
  ["minusdu"] = 10794,
  ["loplus"] = 10797,
  ["roplus"] = 10798,
  ["Cross"] = 10799,
  ["timesd"] = 10800,
  ["timesbar"] = 10801,
  ["smashp"] = 10803,
  ["lotimes"] = 10804,
  ["rotimes"] = 10805,
  ["otimesas"] = 10806,
  ["Otimes"] = 10807,
  ["odiv"] = 10808,
  ["triplus"] = 10809,
  ["triminus"] = 10810,
  ["tritime"] = 10811,
  ["iprod"] = 10812,
  ["intprod"] = 10812,
  ["amalg"] = 10815,
  ["capdot"] = 10816,
  ["ncup"] = 10818,
  ["ncap"] = 10819,
  ["capand"] = 10820,
  ["cupor"] = 10821,
  ["cupcap"] = 10822,
  ["capcup"] = 10823,
  ["cupbrcap"] = 10824,
  ["capbrcup"] = 10825,
  ["cupcup"] = 10826,
  ["capcap"] = 10827,
  ["ccups"] = 10828,
  ["ccaps"] = 10829,
  ["ccupssm"] = 10832,
  ["And"] = 10835,
  ["Or"] = 10836,
  ["andand"] = 10837,
  ["oror"] = 10838,
  ["orslope"] = 10839,
  ["andslope"] = 10840,
  ["andv"] = 10842,
  ["orv"] = 10843,
  ["andd"] = 10844,
  ["ord"] = 10845,
  ["wedbar"] = 10847,
  ["sdote"] = 10854,
  ["simdot"] = 10858,
```

```
["congdot"] = 10861,
["easter"] = 10862,
["apacir"] = 10863,
["apE"] = 10864,
["eplus"] = 10865,
["pluse"] = 10866,
["Esim"] = 10867,
["Colone"] = 10868,
["Equal"] = 10869,
["eDDot"] = 10871,
["ddotseq"] = 10871,
["equivDD"] = 10872,
["ltcir"] = 10873,
["gtcir"] = 10874,
["ltquest"] = 10875,
["gtquest"] = 10876,
["les"] = 10877,
["LessSlantEqual"] = 10877,
["leqslant"] = 10877,
["ges"] = 10878,
["GreaterSlantEqual"] = 10878,
["geqslant"] = 10878,
["lesdot"] = 10879,
["gesdot"] = 10880,
["lesdoto"] = 10881,
["gesdoto"] = 10882,
["lesdotor"] = 10883,
["gesdotol"] = 10884,
["lap"] = 10885,
["lessapprox"] = 10885,
["gap"] = 10886,
["gtrapprox"] = 10886,
["lne"] = 10887,
["lneq"] = 10887,
["gne"] = 10888,
["gneq"] = 10888,
["lnap"] = 10889,
["lnapprox"] = 10889,
["gnap"] = 10890,
["gnapprox"] = 10890,
["lEg"] = 10891,
["lesseqqgtr"] = 10891,
["gEl"] = 10892,
["gtreqqless"] = 10892,
["lsime"] = 10893,
["gsime"] = 10894,
["lsimg"] = 10895,
```

```
  ["gsiml"] = 10896,
  ["lgE"] = 10897,
  ["glE"] = 10898,
  ["lesges"] = 10899,
  ["gesles"] = 10900,
  ["els"] = 10901,
  ["eqslantless"] = 10901,
  ["egs"] = 10902,
  ["eqslantgtr"] = 10902,
  ["elsdot"] = 10903,
  ["egsdot"] = 10904,
  ["el"] = 10905,
  ["eg"] = 10906,
  ["siml"] = 10909,
  ["simg"] = 10910,
  ["simlE"] = 10911,
  ["simgE"] = 10912,
  ["LessLess"] = 10913,
  ["GreaterGreater"] = 10914,
  ["glj"] = 10916,
  ["gla"] = 10917,
  ["ltcc"] = 10918,
  ["gtcc"] = 10919,
  ["lescc"] = 10920,
  ["gescc"] = 10921,
  ["smt"] = 10922,
  ["lat"] = 10923,
  ["smte"] = 10924,
  ["late"] = 10925,
  ["bumpE"] = 10926,
  ["pre"] = 10927,
  ["preceq"] = 10927,
  ["PrecedesEqual"] = 10927,
  ["sce"] = 10928,
  ["succeq"] = 10928,
  ["SucceedsEqual"] = 10928,
  ["prE"] = 10931,
  ["scE"] = 10932,
  ["prnE"] = 10933,
  ["precneqq"] = 10933,
  ["scnE"] = 10934,
  ["succneqq"] = 10934,
  ["prap"] = 10935,
  ["precapprox"] = 10935,
  ["scap"] = 10936,
  ["succapprox"] = 10936,
  ["prnap"] = 10937,
```

```
["precnapprox"] = 10937,
["scnap"] = 10938,
["succnapprox"] = 10938,
["Pr"] = 10939,
["Sc"] = 10940,
["subdot"] = 10941,
["supdot"] = 10942,
["subplus"] = 10943,
["supplus"] = 10944,
["submult"] = 10945,
["supmult"] = 10946,
["subedot"] = 10947,
["supedot"] = 10948,
["subE"] = 10949,
["subseteqq"] = 10949,
["supE"] = 10950,
["supseteqq"] = 10950,
["subsim"] = 10951,
["supsim"] = 10952,
["subnE"] = 10955,
["subsetneqq"] = 10955,
["supnE"] = 10956,
["supsetneqq"] = 10956,
["csub"] = 10959,
["csup"] = 10960,
["csube"] = 10961,
["csupe"] = 10962,
["subsup"] = 10963,
["supsub"] = 10964,
["subsub"] = 10965,
["supsup"] = 10966,
["suphsub"] = 10967,
["supdsub"] = 10968,
["forkv"] = 10969,
["topfork"] = 10970,
["mlcp"] = 10971,
["Dashv"] = 10980,
["DoubleLeftTee"] = 10980,
["Vdashl"] = 10982,
["Barv"] = 10983,
["vBar"] = 10984,
["vBarv"] = 10985,
["Vbar"] = 10987,
["Not"] = 10988,
["bNot"] = 10989,
["rnmid"] = 10990,
["cirmid"] = 10991,
```

```
4318  ["midcir"] = 10992,
4319  ["topcir"] = 10993,
4320  ["nhpar"] = 10994,
4321  ["parsim"] = 10995,
4322  ["parsl"] = 11005,
4323  ["fflig"] = 64256,
4324  ["filig"] = 64257,
4325  ["fllig"] = 64258,
4326  ["ffilig"] = 64259,
4327  ["ffllig"] = 64260,
4328  ["Ascr"] = 119964,
4329  ["Cscr"] = 119966,
4330  ["Dscr"] = 119967,
4331  ["Gscr"] = 119970,
4332  ["Jscr"] = 119973,
4333  ["Kscr"] = 119974,
4334  ["Nscr"] = 119977,
4335  ["Oscr"] = 119978,
4336  ["Pscr"] = 119979,
4337  ["Qscr"] = 119980,
4338  ["Sscr"] = 119982,
4339  ["Tscr"] = 119983,
4340  ["Uscr"] = 119984,
4341  ["Vscr"] = 119985,
4342  ["Wscr"] = 119986,
4343  ["Xscr"] = 119987,
4344  ["Yscr"] = 119988,
4345  ["Zscr"] = 119989,
4346  ["ascr"] = 119990,
4347  ["bscr"] = 119991,
4348  ["cscr"] = 119992,
4349  ["dscr"] = 119993,
4350  ["fscr"] = 119995,
4351  ["hscr"] = 119997,
4352  ["iscr"] = 119998,
4353  ["jscr"] = 119999,
4354  ["kscr"] = 120000,
4355  ["lscr"] = 120001,
4356  ["mscr"] = 120002,
4357  ["nscr"] = 120003,
4358  ["pscr"] = 120005,
4359  ["qscr"] = 120006,
4360  ["rscr"] = 120007,
4361  ["sscr"] = 120008,
4362  ["tscr"] = 120009,
4363  ["uscr"] = 120010,
4364  ["vscr"] = 120011,
```

```
4365   ["wscr"] = 120012,
4366   ["xscr"] = 120013,
4367   ["yscr"] = 120014,
4368   ["zscr"] = 120015,
4369   ["Afr"] = 120068,
4370   ["Bfr"] = 120069,
4371   ["Dfr"] = 120071,
4372   ["Efr"] = 120072,
4373   ["Ffr"] = 120073,
4374   ["Gfr"] = 120074,
4375   ["Jfr"] = 120077,
4376   ["Kfr"] = 120078,
4377   ["Lfr"] = 120079,
4378   ["Mfr"] = 120080,
4379   ["Nfr"] = 120081,
4380   ["Ofr"] = 120082,
4381   ["Pfr"] = 120083,
4382   ["Qfr"] = 120084,
4383   ["Sfr"] = 120086,
4384   ["Tfr"] = 120087,
4385   ["Ufr"] = 120088,
4386   ["Vfr"] = 120089,
4387   ["Wfr"] = 120090,
4388   ["Xfr"] = 120091,
4389   ["Yfr"] = 120092,
4390   ["afr"] = 120094,
4391   ["bfr"] = 120095,
4392   ["cfr"] = 120096,
4393   ["dfr"] = 120097,
4394   ["efr"] = 120098,
4395   ["ffr"] = 120099,
4396   ["gfr"] = 120100,
4397   ["hfr"] = 120101,
4398   ["ifr"] = 120102,
4399   ["jfr"] = 120103,
4400   ["kfr"] = 120104,
4401   ["lfr"] = 120105,
4402   ["mfr"] = 120106,
4403   ["nfr"] = 120107,
4404   ["ofr"] = 120108,
4405   ["pfr"] = 120109,
4406   ["qfr"] = 120110,
4407   ["rfr"] = 120111,
4408   ["sfr"] = 120112,
4409   ["tfr"] = 120113,
4410   ["ufr"] = 120114,
4411   ["vfr"] = 120115,
```

```
4412  ["wfr"] = 120116,
4413  ["xfr"] = 120117,
4414  ["yfr"] = 120118,
4415  ["zfr"] = 120119,
4416  ["Aopf"] = 120120,
4417  ["Bopf"] = 120121,
4418  ["Dopf"] = 120123,
4419  ["Eopf"] = 120124,
4420  ["Fopf"] = 120125,
4421  ["Gopf"] = 120126,
4422  ["Iopf"] = 120128,
4423  ["Jopf"] = 120129,
4424  ["Kopf"] = 120130,
4425  ["Lopf"] = 120131,
4426  ["Mopf"] = 120132,
4427  ["Oopf"] = 120134,
4428  ["Sopf"] = 120138,
4429  ["Topf"] = 120139,
4430  ["Uopf"] = 120140,
4431  ["Vopf"] = 120141,
4432  ["Wopf"] = 120142,
4433  ["Xopf"] = 120143,
4434  ["Yopf"] = 120144,
4435  ["aopf"] = 120146,
4436  ["bopf"] = 120147,
4437  ["copf"] = 120148,
4438  ["dopf"] = 120149,
4439  ["eopf"] = 120150,
4440  ["fopf"] = 120151,
4441  ["gopf"] = 120152,
4442  ["hopf"] = 120153,
4443  ["iopf"] = 120154,
4444  ["jopf"] = 120155,
4445  ["kopf"] = 120156,
4446  ["lopf"] = 120157,
4447  ["mopf"] = 120158,
4448  ["nopf"] = 120159,
4449  ["oopf"] = 120160,
4450  ["popf"] = 120161,
4451  ["qopf"] = 120162,
4452  ["ropf"] = 120163,
4453  ["sopf"] = 120164,
4454  ["topf"] = 120165,
4455  ["uopf"] = 120166,
4456  ["vopf"] = 120167,
4457  ["wopf"] = 120168,
4458  ["xopf"] = 120169,
```

```
4459   ["yopf"] = 120170,
4460   ["zopf"] = 120171,
4461 }
```

Given a string `s` of decimal digits, the `entities.dec_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```
4462 function entities.dec_entity(s)
4463   return unicode.utf8.char(tonumber(s))
4464 end
```

Given a string `s` of hexadecimal digits, the `entities.hex_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```
4465 function entities.hex_entity(s)
4466   return unicode.utf8.char(tonumber("0x"..s))
4467 end
```

Given a character entity name `s` (like `ouml`), the `entities.char_entity` returns the corresponding UTF8-encoded Unicode codepoint.

```
4468 function entities.char_entity(s)
4469   local n = character_entities[s]
4470   if n == nil then
4471     return "&" .. s .. ";"
4472   end
4473   return unicode.utf8.char(n)
4474 end
```

3.1.3 Plain TEX Writer

This section documents the `writer` object, which implements the routines for producing the TEX output. The object is an amalgamate of the generic, TEX, LATEX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

Although not specified in the Lua interface (see Section 2.1), the `writer` object is exported, so that the curious user could easily tinker with the methods of the objects produced by the `writer.new` method described below. The user should be aware, however, that the implementation may change in a future revision.

```
4475 M.writer = {}
```

The `writer.new` method creates and returns a new TEX writer object associated with the Lua interface options (see Section 2.1.3) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `writer->`⟨*member*⟩. All member variables are immutable unless explicitly stated otherwise.

```
function M.writer.new(options)
  local self = {}
```

Make `options` available as `writer->options`, so that it is accessible from extensions.

```
  self.options = options
```

Parse the `slice` option and define `writer->slice_begin`, `writer->slice_end`, and `writer->is_writing`. The `writer->is_writing` member variable is mutable.

```
  local slice_specifiers = {}
  for specifier in options.slice:gmatch("[^%s]+") do
    table.insert(slice_specifiers, specifier)
  end

  if #slice_specifiers == 2 then
    self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
    local slice_begin_type = self.slice_begin:sub(1, 1)
    if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
      self.slice_begin = "^" .. self.slice_begin
    end
    local slice_end_type = self.slice_end:sub(1, 1)
    if slice_end_type ~= "^" and slice_end_type ~= "$" then
      self.slice_end = "$" .. self.slice_end
    end
  elseif #slice_specifiers == 1 then
    self.slice_begin = "^" .. slice_specifiers[1]
    self.slice_end = "$" .. slice_specifiers[1]
  end

  if self.slice_begin == "^" and self.slice_end ~= "^" then
    self.is_writing = true
  else
    self.is_writing = false
  end
```

Define `writer->suffix` as the suffix of the produced cache files.

```
  self.suffix = ".tex"
```

Define `writer->space` as the output format of a space character.

```
  self.space = " "
```

Define `writer->nbsp` as the output format of a non-breaking space character.

```
  self.nbsp = "\\markdownRendererNbsp{}"
```

Define `writer->plain` as a function that will transform an input plain text block `s` to the output format.

```
  function self.plain(s)
    return s
  end
```

Define `writer->paragraph` as a function that will transform an input paragraph `s` to the output format.

```
4510  function self.paragraph(s)
4511    if not self.is_writing then return "" end
4512    return s
4513  end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
4514  function self.pack(name)
4515    return [[\input ]] .. name .. [[\relax]]
4516  end
```

Define `writer->interblocksep` as the output format of a block element separator.

```
4517  function self.interblocksep()
4518    if not self.is_writing then return "" end
4519    return "\\markdownRendererInterblockSeparator\n{}"
4520  end
```

Define `writer->linebreak` as the output format of a forced line break.

```
4521  self.linebreak = "\\markdownRendererLineBreak\n{}"
```

Define `writer->ellipsis` as the output format of an ellipsis.

```
4522  self.ellipsis = "\\markdownRendererEllipsis{}"
```

Define `writer->hrule` as the output format of a horizontal rule.

```
4523  function self.hrule()
4524    if not self.is_writing then return "" end
4525    return "\\markdownRendererHorizontalRule{}"
4526  end
```

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```
4527   self.escaped_uri_chars = {
4528     ["{"] = "\\markdownRendererLeftBrace{}",
4529     ["}"] = "\\markdownRendererRightBrace{}",
4530     ["%"] = "\\markdownRendererPercentSign{}",
4531     ["\\"] = "\\markdownRendererBackslash{}",
4532   }
4533   self.escaped_minimal_strings = {
4534     ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4535     ["☒"] = "\\markdownRendererTickedBox{}",
4536     ["⊡"] = "\\markdownRendererHalfTickedBox{}",
4537     ["▫"] = "\\markdownRendererUntickedBox{}",
4538   }
```

Define a table `writer->escaped_chars` containing the mapping from special plain TEX characters (including the active pipe character (`|`) of ConTEXt) that need to be escaped for typeset content.

```
  self.escaped_chars = {
     ["{"] = "\\markdownRendererLeftBrace{}",
     ["}"] = "\\markdownRendererRightBrace{}",
     ["%"] = "\\markdownRendererPercentSign{}",
     ["\\"] = "\\markdownRendererBackslash{}",
     ["#"] = "\\markdownRendererHash{}",
     ["$"] = "\\markdownRendererDollarSign{}",
     ["&"] = "\\markdownRendererAmpersand{}",
     ["_"] = "\\markdownRendererUnderscore{}",
     ["^"] = "\\markdownRendererCircumflex{}",
     ["~"] = "\\markdownRendererTilde{}",
     ["|"] = "\\markdownRendererPipe{}",
   }
```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minima` tables to create the `writer->escape`, `writer->escape_uri`, and `writer->escape_minimal` escaper functions.

```
  self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
  self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
  self.escape_minimal = util.escaper({}, self.escaped_minimal_strings)
```

Define `writer->string` as a function that will transform an input plain text span `s` to the output format and `writer->uri` as a function that will transform an input URI `u` to the output format. If the `hybrid` option is enabled, use the `writer->escape_minimal`. Otherwise, use the `writer->escape`, and `writer->escape_uri` functions.

```
  if options.hybrid then
    self.string = self.escape_minimal
    self.uri = self.escape_minimal
  else
    self.string = self.escape
    self.uri = self.escape_uri
  end
```

Define `writer->code` as a function that will transform an input inlined code span `s` to the output format.

```
  function self.code(s)
    return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
  end
```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, and `tit` to the title of the link.

```
  function self.link(lab,src,tit)
    return {"\\markdownRendererLink{",lab,"}",
                          "{",self.escape(src),"}",
                          "{",self.uri(src),"}",
                          "{",self.string(tit or ""),"}"}
```

```
4570    end
```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, and `tit` to the title of the image.

```
4571    function self.image(lab,src,tit)
4572      return {"\\markdownRendererImage{",lab,"}",
4573                               "{",self.string(src),"}",
4574                               "{",self.uri(src),"}",
4575                               "{",self.string(tit or ""),"}"}
4576    end
```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```
4577    function self.bulletlist(items,tight)
4578      if not self.is_writing then return "" end
4579      local buffer = {}
4580      for _,item in ipairs(items) do
4581        buffer[#buffer + 1] = self.bulletitem(item)
4582      end
4583      local contents = util.intersperse(buffer,"\n")
4584      if tight and options.tightLists then
4585        return {"\\markdownRendererUlBeginTight\n",contents,
4586          "\n\\markdownRendererUlEndTight "}
4587      else
4588        return {"\\markdownRendererUlBegin\n",contents,
4589          "\n\\markdownRendererUlEnd "}
4590      end
4591    end
```

Define `writer->bulletitem` as a function that will transform an input bulleted list item to the output format, where `s` is the text of the list item.

```
4592    function self.bulletitem(s)
4593      return {"\\markdownRendererUlItem ",s,
4594              "\\markdownRendererUlItemEnd "}
4595    end
```

Define `writer->orderedlist` as a function that will transform an input ordered list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not. If the optional parameter `startnum` is present, it is the number of the first list item.

```
4596    function self.orderedlist(items,tight,startnum)
4597      if not self.is_writing then return "" end
4598      local buffer = {}
4599      local num = startnum
4600      for _,item in ipairs(items) do
4601        buffer[#buffer + 1] = self.ordereditem(item,num)
```

```
4602        if num ~= nil then
4603          num = num + 1
4604        end
4605      end
4606      local contents = util.intersperse(buffer,"\n")
4607      if tight and options.tightLists then
4608        return {"\\markdownRendererOlBeginTight\n",contents,
4609                "\n\\markdownRendererOlEndTight "}
4610      else
4611        return {"\\markdownRendererOlBegin\n",contents,
4612                "\n\\markdownRendererOlEnd "}
4613      end
4614    end
```

Define `writer->ordereditem` as a function that will transform an input ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```
4615    function self.ordereditem(s,num)
4616      if num ~= nil then
4617        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
4618                "\\markdownRendererOlItemEnd "}
4619      else
4620        return {"\\markdownRendererOlItem ",s,
4621                "\\markdownRendererOlItemEnd "}
4622      end
4623    end
```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```
4624    function self.inline_html_comment(contents)
4625      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
4626    end
```

Define `writer->block_html_comment` as a function that will transform the contents of a block HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```
4627    function self.block_html_comment(contents)
4628      if not self.is_writing then return "" end
4629      return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4630              "\n\\markdownRendererBlockHtmlCommentEnd "}
4631    end
```

Define `writer->inline_html_tag` as a function that will transform the contents of an opening, closing, or empty inline HTML tag to the output format, where `contents` are the contents of the HTML tag.

```
4632    function self.inline_html_tag(contents)
4633      return {"\\markdownRendererInlineHtmlTag{",self.string(contents),"}"}
```

```
4634    end
```

Define `writer->block_html_element` as a function that will transform the contents of a block HTML element to the output format, where `s` are the contents of the HTML element.

```
4635    function self.block_html_element(s)
4636      if not self.is_writing then return "" end
4637      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4638      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
4639    end
```

Define `writer->emphasis` as a function that will transform an emphasized span `s` of input text to the output format.

```
4640    function self.emphasis(s)
4641      return {"\\markdownRendererEmphasis{",s,"}"}
4642    end
```

Define `writer->tickbox` as a function that will transform a number `f` to the output format.

```
4643    function self.tickbox(f)
4644      if f == 1.0 then
4645        return "☒ "
4646      elseif f == 0.0 then
4647        return "☐ "
4648      else
4649        return "☒ "
4650      end
4651    end
```

Define `writer->strong` as a function that will transform a strongly emphasized span `s` of input text to the output format.

```
4652    function self.strong(s)
4653      return {"\\markdownRendererStrongEmphasis{",s,"}"}
4654    end
```

Define `writer->blockquote` as a function that will transform an input block quote `s` to the output format.

```
4655    function self.blockquote(s)
4656      if #util.rope_to_string(s) == 0 then return "" end
4657      return {"\\markdownRendererBlockQuoteBegin\n",s,
4658        "\n\\markdownRendererBlockQuoteEnd "}
4659    end
```

Define `writer->verbatim` as a function that will transform an input code block `s` to the output format.

```
4660    function self.verbatim(s)
4661      if not self.is_writing then return "" end
4662      s = string.gsub(s, '[\r\n%s]*$', '')
4663      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
```

```
4664    return {"\\markdownRendererInputVerbatim{",name,"}"}
4665  end
```

Define `writer->document` as a function that will transform a document `d` to the output format.

```
4666  function self.document(d)
4667    local active_attributes = self.active_attributes
4668    local buf = {"\\markdownRendererDocumentBegin\n", d}
4669
4670    -- pop attributes for sections that have ended
4671    if options.headerAttributes and self.is_writing then
4672      while #active_attributes > 0 do
4673        local attributes = active_attributes[#active_attributes]
4674        if #attributes > 0 then
4675          table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4676        end
4677        table.remove(active_attributes, #active_attributes)
4678      end
4679    end
4680
4681    table.insert(buf, "\\markdownRendererDocumentEnd")
4682
4683    return buf
4684  end
```

Define `writer->active_attributes` as a stack of attributes of the headings that are currently active. The `writer->active_headings` member variable is mutable.

```
4685  self.active_attributes = {}
```

Define `writer->heading` as a function that will transform an input heading `s` at level `level` with identifiers `identifiers` to the output format.

```
4686  function self.heading(s, level, attributes)
4687    attributes = attributes or {}
4688    for i = 1, #attributes do
4689      attributes[attributes[i]] = true
4690    end
4691
4692    local active_attributes = self.active_attributes
4693    local slice_begin_type = self.slice_begin:sub(1, 1)
4694    local slice_begin_identifier = self.slice_begin:sub(2) or ""
4695    local slice_end_type = self.slice_end:sub(1, 1)
4696    local slice_end_identifier = self.slice_end:sub(2) or ""
4697
4698    local buf = {}
4699
4700    -- push empty attributes for implied sections
4701    while #active_attributes < level-1 do
4702      table.insert(active_attributes, {})
```

```
4703    end
4704
4705    -- pop attributes for sections that have ended
4706    while #active_attributes >= level do
4707      local active_identifiers = active_attributes[#active_attributes]
4708      -- tear down all active attributes at slice end
4709      if active_identifiers["#" .. slice_end_identifier] ~= nil
4710          and slice_end_type == "$" then
4711        for header_level = #active_attributes, 1, -1 do
4712          if options.headerAttributes and #active_attributes[header_level] > 0 then
4713            table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4714          end
4715        end
4716        self.is_writing = false
4717      end
4718      table.remove(active_attributes, #active_attributes)
4719      if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4720        table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4721      end
4722      -- apply all active attributes at slice beginning
4723      if active_identifiers["#" .. slice_begin_identifier] ~= nil
4724          and slice_begin_type == "$" then
4725        for header_level = 1, #active_attributes do
4726          if options.headerAttributes and #active_attributes[header_level] > 0 then
4727            table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4728          end
4729        end
4730        self.is_writing = true
4731      end
4732    end
4733
4734    -- tear down all active attributes at slice end
4735    if attributes["#" .. slice_end_identifier] ~= nil
4736        and slice_end_type == "^" then
4737      for header_level = #active_attributes, 1, -1 do
4738        if options.headerAttributes and #active_attributes[header_level] > 0 then
4739          table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4740        end
4741      end
4742      self.is_writing = false
4743    end
4744
4745    -- push attributes for the new section
4746    table.insert(active_attributes, attributes)
4747    if self.is_writing and options.headerAttributes and #attributes > 0 then
4748      table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4749    end
```

```
4750
4751      -- apply all active attributes at slice beginning
4752      if attributes["#" .. slice_begin_identifier] ~= nil
4753          and slice_begin_type == "^" then
4754        for header_level = 1, #active_attributes do
4755          if options.headerAttributes and #active_attributes[header_level] > 0 then
4756            table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4757          end
4758        end
4759        self.is_writing = true
4760      end
4761
4762      if self.is_writing then
4763        table.sort(attributes)
4764        local key, value
4765        for i = 1, #attributes do
4766          if attributes[i]:sub(1, 1) == "#" then
4767            table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
4768                               attributes[i]:sub(2), "}"})
4769          elseif attributes[i]:sub(1, 1) == "." then
4770            table.insert(buf, {"\\markdownRendererAttributeClassName{",
4771                               attributes[i]:sub(2), "}"})
4772          else
4773            key, value = attributes[i]:match("([^= ]+)%s*=%s*(.*)")
4774            table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
4775                               key, "}{", value, "}"})
4776          end
4777        end
4778      end
4779
4780      local cmd
4781      level = level + options.shiftHeadings
4782      if level <= 1 then
4783        cmd = "\\markdownRendererHeadingOne"
4784      elseif level == 2 then
4785        cmd = "\\markdownRendererHeadingTwo"
4786      elseif level == 3 then
4787        cmd = "\\markdownRendererHeadingThree"
4788      elseif level == 4 then
4789        cmd = "\\markdownRendererHeadingFour"
4790      elseif level == 5 then
4791        cmd = "\\markdownRendererHeadingFive"
4792      elseif level >= 6 then
4793        cmd = "\\markdownRendererHeadingSix"
4794      else
4795        cmd = ""
4796      end
```

```
4797      if self.is_writing then
4798        table.insert(buf, {cmd, "{", s, "}"})
4799      end
4800
4801      return buf
4802    end
```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```
4803    function self.get_state()
4804      return {
4805        is_writing=self.is_writing,
4806        active_attributes={table.unpack(self.active_attributes)},
4807      }
4808    end
```

Define `writer->set_state` as a function that restores the input state `s` and returns the previous state of the writer.

```
4809    function self.set_state(s)
4810      local previous_state = self.get_state()
4811      for key, value in pairs(s) do
4812        self[key] = value
4813      end
4814      return previous_state
4815    end
```

Define `writer->defer_call` as a function that will encapsulate the input function `f`, so that `f` is called with the state of the writer at the time of calling `writer->defer_call`.

```
4816    function self.defer_call(f)
4817      local previous_state = self.get_state()
4818      return function(...)
4819        local state = self.set_state(previous_state)
4820        local return_value = f(...)
4821        self.set_state(state)
4822        return return_value
4823      end
4824    end
4825
4826    return self
4827  end
```

3.1.4 Parsers

The `parsers` hash table stores PEG patterns that are static and can be reused between different `reader` objects.

```
4828  local parsers                  = {}
```

3.1.4.1 Basic Parsers

```
parsers.percent             = P("%")
parsers.at                  = P("@")
parsers.comma               = P(",")
parsers.asterisk            = P("*")
parsers.dash                = P("-")
parsers.plus                = P("+")
parsers.underscore          = P("_")
parsers.period              = P(".")
parsers.hash                = P("#")
parsers.ampersand           = P("&")
parsers.backtick            = P("`")
parsers.less                = P("<")
parsers.more                = P(">")
parsers.space               = P(" ")
parsers.squote              = P("'")
parsers.dquote              = P('"')
parsers.lparent             = P("(")
parsers.rparent             = P(")")
parsers.lbracket            = P("[")
parsers.rbracket            = P("]")
parsers.lbrace              = P("{")
parsers.rbrace              = P("}")
parsers.circumflex          = P("^")
parsers.slash               = P("/")
parsers.equal               = P("=")
parsers.colon               = P(":")
parsers.semicolon           = P(";")
parsers.exclamation         = P("!")
parsers.pipe                = P("|")
parsers.tilde               = P("~")
parsers.backslash           = P("\\")
parsers.tab                 = P("\t")
parsers.newline             = P("\n")
parsers.tightblocksep       = P("\001")

parsers.digit               = R("09")
parsers.hexdigit            = R("09","af","AF")
parsers.letter              = R("AZ","az")
parsers.alphanumeric        = R("AZ","az","09")
parsers.keyword             = parsers.letter
                            * parsers.alphanumeric^0
parsers.internal_punctuation = S(":;,.?")

parsers.doubleasterisks     = P("**")
parsers.doubleunderscores   = P("__")
parsers.doubletildes        = P("~~")
```

```
parsers.fourspaces              = P("    ")

parsers.any                     = P(1)
parsers.fail                    = parsers.any - 1

parsers.escapable               = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
parsers.anyescaped              = parsers.backslash / "" * parsers.escapable
                                + parsers.any

parsers.spacechar               = S("\t ")
parsers.spacing                 = S(" \n\r\t")
parsers.nonspacechar            = parsers.any - parsers.spacing
parsers.optionalspace           = parsers.spacechar^0

parsers.normalchar              = parsers.any - (V("SpecialChar")
                                              + parsers.spacing
                                              + parsers.tightblocksep)
parsers.eof                     = -parsers.any
parsers.nonindentspace          = parsers.space^-3 * - parsers.spacechar
parsers.indent                  = parsers.space^-3 * parsers.tab
                                + parsers.fourspaces / ""
parsers.linechar                = P(1 - parsers.newline)

parsers.blankline               = parsers.optionalspace
                                * parsers.newline / "\n"
parsers.blanklines              = parsers.blankline^0
parsers.skipblanklines          = (parsers.optionalspace * parsers.newline)^0
parsers.indentedline            = parsers.indent    /""
                                * C(parsers.linechar^1 * parsers.newline^-
1)
parsers.optionallyindentedline = parsers.indent^-1 /""
                                * C(parsers.linechar^1 * parsers.newline^-
1)
parsers.sp                      = parsers.spacing^0
parsers.spnl                    = parsers.optionalspace
                                * (parsers.newline * parsers.optionalspace)^-
1
parsers.line                    = parsers.linechar^0 * parsers.newline
parsers.nonemptyline            = parsers.line - parsers.blankline
```

The `parsers.commented_line^1` parser recognizes the regular language of TEX comments, see an equivalent finite automaton in Figure 6.

```
parsers.commented_line_letter  = parsers.linechar
                               + parsers.newline
                               - parsers.backslash
                               - parsers.percent
parsers.commented_line         = Cg(Cc(""), "backslashes")
```

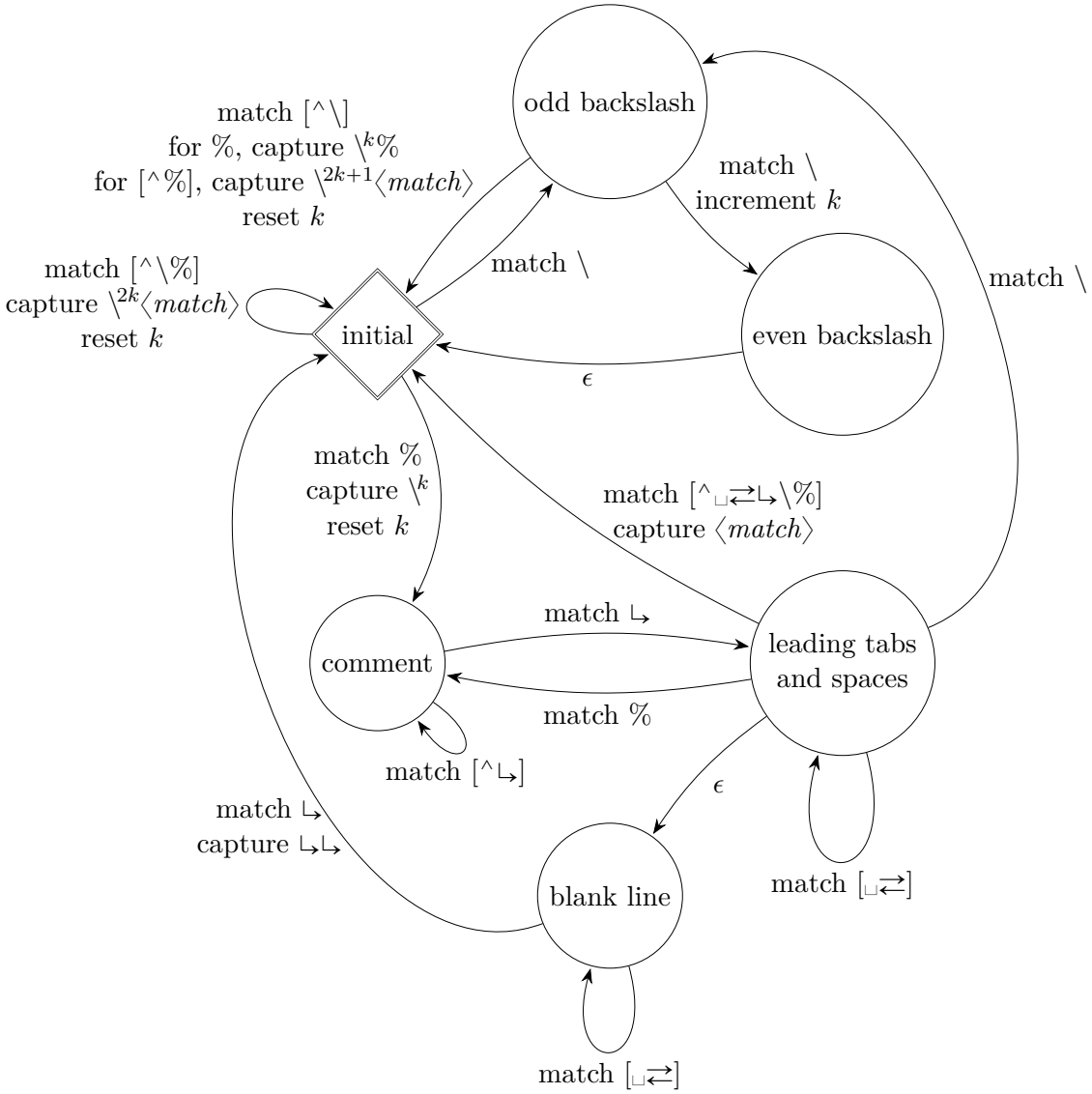

Figure 6: A pushdown automaton that recognizes TEX comments

```
                                * ((#(parsers.commented_line_letter
                                    - parsers.newline)
                                   * Cb("backslashes")
                                   * Cs(parsers.commented_line_letter
                                     - parsers.newline)^1  -- initial
                                   * Cg(Cc(""), "backslashes"))
                                  + #(parsers.backslash * parsers.backslash)
                                  * Cg((parsers.backslash  -- even backslash
                                       * parsers.backslash)^1, "backslashes")
                                  + (parsers.backslash
                                    * (#parsers.percent
                                      * Cb("backslashes")
                                      / function(backslashes)
                                        return string.rep("\\", #backslashes / 2)
                                      end
                                      * C(parsers.percent)
                                      + #parsers.commented_line_letter
                                      * Cb("backslashes")
                                      * Cc("\\")
                                      * C(parsers.commented_line_letter))
                                    * Cg(Cc(""), "backslashes")))^0
                                * (#parsers.percent
                                  * Cb("backslashes")
                                  / function(backslashes)
                                    return string.rep("\\", #backslashes / 2)
                                  end
                                  * ((parsers.percent  -- comment
                                     * parsers.line
                                     * #parsers.blankline) -- blank line
                                    / "\n"
                                    + parsers.percent  -- comment
                                    * parsers.line
                                    * parsers.optionalspace)  -- leading tabs and spaces
                                  + #(parsers.newline)
                                  * Cb("backslashes")
                                  * C(parsers.newline))

parsers.chunk                   = parsers.line * (parsers.optionallyindentedline
                                                  - parsers.blankline)^0

parsers.attribute_key_char      = parsers.alphanumeric + S("_-")
parsers.attribute_key           = (parsers.attribute_key_char
                                  - parsers.dash - parsers.digit)
                                * parsers.attribute_key_char^0
parsers.attribute_value         = ( (parsers.dquote / "")
                                  * (parsers.anyescaped - parsers.dquote)^0
                                  * (parsers.dquote / ""))
```

```
                                  + ( parsers.anyescaped - parsers.dquote - parsers.rbrace
                                    - parsers.space)^0

-- block followed by 0 or more optionally
-- indented blocks with first line indented.
parsers.indented_blocks = function(bl)
  return Cs( bl
         * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
         * (parsers.blankline^1 + parsers.eof) )
end
```

3.1.4.2 Parsers Used for Markdown Lists

```
parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)

parsers.bullet = ( parsers.bulletchar * #parsers.spacing
                                      * (parsers.tab + parsers.space^-3)
                 + parsers.space * parsers.bulletchar * #parsers.spacing
                                 * (parsers.tab + parsers.space^-2)
                 + parsers.space * parsers.space * parsers.bulletchar
                                 * #parsers.spacing
                                 * (parsers.tab + parsers.space^-1)
                 + parsers.space * parsers.space * parsers.space
                                 * parsers.bulletchar * #parsers.spacing
                 )

local function tickbox(interior)
  return parsers.optionalspace * parsers.lbracket
       * interior * parsers.rbracket * parsers.spacechar^1
end

parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)

```

3.1.4.3 Parsers Used for Markdown Code Spans

```
parsers.openticks   = Cg(parsers.backtick^1, "ticks")

local function captures_equal_length(_,i,a,b)
  return #a == #b and i
end

parsers.closeticks  = parsers.space^-1
                    * Cmt(C(parsers.backtick^1)
                         * Cb("ticks"), captures_equal_length)

```

```
parsers.intickschar = (parsers.any - S(" \n\r`"))
                    + (parsers.newline * -parsers.blankline)
                    + (parsers.space - parsers.closeticks)
                    + (parsers.backtick^1 - parsers.closeticks)

parsers.inticks     = parsers.openticks * parsers.space^-1
                    * C(parsers.intickschar^0) * parsers.closeticks
```

3.1.4.4 Parsers Used for Fenced Code Blocks

```
local function captures_geq_length(_,i,a,b)
  return #a >= #b and i
end

parsers.infostring     = (parsers.linechar - (parsers.backtick
                       + parsers.space^1 * (parsers.newline + parsers.eof)))^0

local fenceindent
parsers.fencehead    = function(char)
  return               C(parsers.nonindentspace) / function(s) fenceindent = #s end
                     * Cg(char^3, "fencelength")
                     * parsers.optionalspace * C(parsers.infostring)
                     * parsers.optionalspace * (parsers.newline + parsers.eof)
end

parsers.fencetail    = function(char)
  return               parsers.nonindentspace
                     * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
                     * parsers.optionalspace * (parsers.newline + parsers.eof)
                     + parsers.eof
end

parsers.fencedline   = function(char)
  return               C(parsers.line - parsers.fencetail(char))
                     / function(s)
                         local i = 1
                         local remaining = fenceindent
                         while true do
                           local c = s:sub(i, i)
                           if c == " " and remaining > 0 then
                             remaining = remaining - 1
                             i = i + 1
                           elseif c == "\t" and remaining > 3 then
                             remaining = remaining - 4
                             i = i + 1
                           else
                             break
```

```
                            end
                          end
                          return s:sub(i)
                        end
end
```

3.1.4.5 Parsers Used for Markdown Tags and Links

```
parsers.leader      = parsers.space^-3

-- content in balanced brackets, parentheses, or quotes:
parsers.bracketed   = P{ parsers.lbracket
                       * (( parsers.backslash / "" * parsers.rbracket
                          + parsers.any - (parsers.lbracket
                                         + parsers.rbracket
                                         + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.rbracket }

parsers.inparens    = P{ parsers.lparent
                       * ((parsers.anyescaped - (parsers.lparent
                                               + parsers.rparent
                                               + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.rparent }

parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.squote
                                               + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.squote }

parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.dquote
                                               + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.dquote }

-- bracketed tag for markdown links, allowing nested brackets:
parsers.tag         = parsers.lbracket
                    * Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + ( parsers.backslash / "" * parsers.rbracket
                           + parsers.any
                           - (parsers.rbracket + parsers.blankline^2)))^0)
                    * parsers.rbracket
```

```
5093
5094 -- url for markdown links, allowing nested brackets:
5095 parsers.url         = parsers.less * Cs((parsers.anyescaped
5096                                        - parsers.more)^0)
5097                                    * parsers.more
5098                     + Cs((parsers.inparens + (parsers.anyescaped
5099                                               - parsers.spacing
5100                                               - parsers.rparent))^1)
5101
5102 -- quoted text, possibly with nested quotes:
5103 parsers.title_s     = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5104                                           + parsers.squoted)^0)
5105                                     * parsers.squote
5106
5107 parsers.title_d     = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5108                                           + parsers.dquoted)^0)
5109                                     * parsers.dquote
5110
5111 parsers.title_p     = parsers.lparent
5112                     * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5113                     * parsers.rparent
5114
5115 parsers.title       = parsers.title_d + parsers.title_s + parsers.title_p
5116
5117 parsers.optionaltitle
5118                     = parsers.spnl * parsers.title * parsers.spacechar^0
5119                     + Cc("")
```

3.1.4.6 Parsers Used for HTML

```
5120 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5121 parsers.keyword_exact = function(s)
5122   local parser = P(0)
5123   for i=1,#s do
5124     local c = s:sub(i,i)
5125     local m = c .. upper(c)
5126     parser = parser * S(m)
5127   end
5128   return parser
5129 end
5130
5131 parsers.block_keyword =
5132     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5133     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5134     parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5135     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5136     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
```

```
    parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
    parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
    parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
    parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
    parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
    parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
    parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
    parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
    parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
    parsers.keyword_exact("td") + parsers.keyword_exact("tr")

-- There is no reason to support bad html, so we expect quoted attributes
parsers.htmlattributevalue
                          = parsers.squote * (parsers.any - (parsers.blankline
                                                           + parsers.squote))^0
                                           * parsers.squote
                          + parsers.dquote * (parsers.any - (parsers.blankline
                                                           + parsers.dquote))^0
                                           * parsers.dquote

parsers.htmlattribute     = parsers.spacing^1
                          * (parsers.alphanumeric + S("_-"))^1
                          * parsers.sp * parsers.equal * parsers.sp
                          * parsers.htmlattributevalue

parsers.htmlcomment       = P("<!--")
                          * parsers.optionalspace
                          * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
                          * parsers.optionalspace
                          * P("-->")

parsers.htmlinstruction   = P("<?") * (parsers.any - P("?>"))^0 * P("?>")

parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
                    * parsers.sp * parsers.more

parsers.openelt_exact = function(s)
  return parsers.less * parsers.sp * parsers.keyword_exact(s)
       * parsers.htmlattribute^0 * parsers.sp * parsers.more
end

parsers.openelt_block = parsers.sp * parsers.block_keyword
                      * parsers.htmlattribute^0 * parsers.sp * parsers.more

```

```
parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
                     * parsers.keyword * parsers.sp * parsers.more

parsers.closeelt_exact = function(s)
  return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
       * parsers.sp * parsers.more
end

parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
                     * parsers.htmlattribute^0 * parsers.sp * parsers.slash
                     * parsers.more

parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
                       * parsers.htmlattribute^0 * parsers.sp * parsers.slash
                       * parsers.more

parsers.displaytext = (parsers.any - parsers.less)^1

-- return content between two matched HTML tags
parsers.in_matched = function(s)
  return { parsers.openelt_exact(s)
         * (V(1) + parsers.displaytext
           + (parsers.less - parsers.closeelt_exact(s)))^0
         * parsers.closeelt_exact(s) }
end

local function parse_matched_tags(s,pos)
  local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
  return lpeg.match(parsers.in_matched(t),s,pos-1)
end

parsers.in_matched_block_tags = parsers.less
                              * Cmt(#parsers.openelt_block, parse_matched_tags)

```

3.1.4.7 Parsers Used for HTML Entities

```
parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
                  * C(parsers.hexdigit^1) * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                  * C(parsers.digit^1) * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
                  * parsers.semicolon
```

3.1.4.8 Helpers for References

```
-- parse a reference definition:  [foo]: /bar "title"
parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
```

```
                                  * parsers.spacechar^0 * parsers.url
                                  * parsers.optionaltitle * parsers.blankline^1
```

3.1.4.9 Inline Elements

```
parsers.Inline         = V("Inline")
parsers.IndentedInline = V("IndentedInline")

-- parse many p between starter and ender
parsers.between = function(p, starter, ender)
  local ender2 = B(parsers.nonspacechar) * ender
  return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
end

parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more
```

3.1.4.10 Block Elements

```
parsers.lineof = function(c)
    return (parsers.leader * (P(c) * parsers.optionalspace)^3
           * (parsers.newline * parsers.blankline^1
             + parsers.newline^-1 * parsers.eof))
end
```

3.1.4.11 Headings

```
parsers.heading_attribute = (parsers.dash * Cc(".unnumbered"))
                          + C((parsers.hash + parsers.period)
                             * parsers.attribute_key)
                          + Cs( parsers.attribute_key
                              * parsers.optionalspace * parsers.equal * parsers.optional
                              * parsers.attribute_value)
parsers.HeadingAttributes = parsers.lbrace
                          * parsers.optionalspace
                          * parsers.heading_attribute
                          * (parsers.spacechar^1
                            * parsers.heading_attribute)^0
                          * parsers.optionalspace
                          * parsers.rbrace

-- parse Atx heading start and return level
parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
                     * -parsers.hash / length

-- parse setext header ending and return level
parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)

local function strip_atx_end(s)
```

```
5265   return s:gsub("[#%s]*\n$","")
5266 end
```

3.1.5 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

The `reader.new` method creates and returns a new TEX reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `reader->`⟨*member*⟩.

```
5267 M.reader = {}
5268 function M.reader.new(writer, options)
5269   local self = {}
```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```
5270   self.writer = writer
5271   self.options = options
```

Create a `reader->parsers` hash table that stores PEG patterns that depend on the received `options`. Make `reader->parsers` inherit from the global `parsers` table.

```
5272   self.parsers = {}
5273   (function(parsers)
5274     setmetatable(self.parsers, {
5275       __index = function (_, key)
5276         return parsers[key]
5277       end
5278     })
5279   end)(parsers)
```

Make `reader->parsers` available as a local `parsers` variable that will shadow the global `parsers` table and will make `reader->parsers` easier to type in the rest of the reader code.

```
5280   local parsers = self.parsers
```

3.1.5.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
5281   function self.normalize_tag(tag)
5282     return string.lower(
```

```
      gsub(util.rope_to_string(tag), "[ \n\r\t]+", " "))
end
```

Define `iterlines` as a function that iterates over the lines of the input string `s`, transforms them using an input function `f`, and reassembles them into a new string, which it returns.

```
local function iterlines(s, f)
  local rope = lpeg.match(Ct((parsers.line / f)^1), s)
  return util.rope_to_string(rope)
end
```

Define `expandtabs` either as an identity function, when the `preserveTabs` Lua interface option is enabled, or to a function that expands tabs into spaces otherwise.

```
if options.preserveTabs then
  self.expandtabs = function(s) return s end
else
  self.expandtabs = function(s)
                      if s:find("\t") then
                        return iterlines(s, util.expand_tabs_in_line)
                      else
                        return s
                      end
                    end
end
```

3.1.5.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
self.parser_functions = {}
self.create_parser = function(name, grammar, toplevel)
  self.parser_functions[name] = function(str)
```

If the parser function is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```
    if toplevel and options.stripIndent then
        local min_prefix_length, min_prefix = nil, ''
        str = iterlines(str, function(line)
            if lpeg.match(parsers.nonemptyline, line) == nil then
                return line
            end
            line = util.expand_tabs_in_line(line)
```

```
            local prefix = lpeg.match(C(parsers.optionalspace), line)
            local prefix_length = #prefix
            local is_shorter = min_prefix_length == nil
            is_shorter = is_shorter or prefix_length < min_prefix_length
            if is_shorter then
                min_prefix_length, min_prefix = prefix_length, prefix
            end
            return line
        end)
        str = str:gsub('^' .. min_prefix, '')
    end
```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain TEX comments from the input string `str` together with the trailing newline characters.

```
    if toplevel and (options.texComments or options.hybrid) then
        str = lpeg.match(Ct(parsers.commented_line^1), str)
        str = util.rope_to_string(str)
    end
    local res = lpeg.match(grammar(), str)
    if res == nil then
      error(format("%s failed on:\n%s", name, str:sub(1,20)))
    else
      return res
    end
  end
end

self.create_parser("parse_blocks",
                   function()
                     return parsers.blocks
                   end, true)

self.create_parser("parse_blocks_nested",
                   function()
                     return parsers.blocks_nested
                   end, false)

self.create_parser("parse_inlines",
                   function()
                     return parsers.inlines
                   end, false)

self.create_parser("parse_inlines_no_link",
                   function()
                     return parsers.inlines_no_link
                   end, false)
```

```
5353
5354   self.create_parser("parse_inlines_no_inline_note",
5355                      function()
5356                        return parsers.inlines_no_inline_note
5357                      end, false)
5358
5359   self.create_parser("parse_inlines_no_html",
5360                      function()
5361                        return parsers.inlines_no_html
5362                      end, false)
5363
5364   self.create_parser("parse_inlines_nbsp",
5365                      function()
5366                        return parsers.inlines_nbsp
5367                      end, false)
```

3.1.5.3 Parsers Used for Markdown Lists (local)

```
5368   if options.hashEnumerators then
5369     parsers.dig = parsers.digit + parsers.hash
5370   else
5371     parsers.dig = parsers.digit
5372   end
5373
5374   parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5375                      + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5376                                          * (parsers.tab + parsers.space^1)
5377                      + C(parsers.dig * parsers.period) * #parsers.spacing
5378                                        * (parsers.tab + parsers.space^-2)
5379                      + parsers.space * C(parsers.dig^2 * parsers.period)
5380                                        * #parsers.spacing
5381                      + parsers.space * C(parsers.dig * parsers.period)
5382                                        * #parsers.spacing
5383                                        * (parsers.tab + parsers.space^-1)
5384                      + parsers.space * parsers.space * C(parsers.dig^1
5385                                        * parsers.period) * #parsers.spacing
```

3.1.5.4 Parsers Used for Blockquotes (local)

```
5386   -- strip off leading > and indents, and run through blocks
5387   parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
    1)/""
5388                              * parsers.linechar^0 * parsers.newline)^1
5389                            * (-(parsers.leader * parsers.more
5390                                + parsers.blankline) * parsers.linechar^1
5391                              * parsers.newline)^0
5392
5393   if not options.breakableBlockquotes then
```

```
5394    parsers.blockquote_body = parsers.blockquote_body
5395                            * (parsers.blankline^0 / "")
5396  end
```

3.1.5.5 Parsers Used for Footnotes (local)

3.1.5.6 Helpers for Links and References (local)

```
5397  -- List of references defined in the document
5398  local references
5399
5400  -- add a reference to the list
5401  local function register_link(tag,url,title)
5402      references[self.normalize_tag(tag)] = { url = url, title = title }
5403      return ""
5404  end
5405
5406  -- lookup link reference and return either
5407  -- the link or nil and fallback text.
5408  local function lookup_reference(label,sps,tag)
5409      local tagpart
5410      if not tag then
5411          tag = label
5412          tagpart = ""
5413      elseif tag == "" then
5414          tag = label
5415          tagpart = "[]"
5416      else
5417          tagpart = {"[",
5418            self.parser_functions.parse_inlines(tag),
5419            "]"}
5420      end
5421      if sps then
5422        tagpart = {sps, tagpart}
5423      end
5424      local r = references[self.normalize_tag(tag)]
5425      if r then
5426        return r
5427      else
5428        return nil, {"[",
5429          self.parser_functions.parse_inlines(label),
5430          "]", tagpart}
5431      end
5432  end
5433
5434  -- lookup link reference and return a link, if the reference is found,
5435  -- or a bracketed label otherwise.
```

```
5436  local function indirect_link(label,sps,tag)
5437    return writer.defer_call(function()
5438      local r,fallback = lookup_reference(label,sps,tag)
5439      if r then
5440        return writer.link(
5441          self.parser_functions.parse_inlines_no_link(label),
5442          r.url, r.title)
5443      else
5444        return fallback
5445      end
5446    end)
5447  end
5448
5449  -- lookup image reference and return an image, if the reference is found,
5450  -- or a bracketed label otherwise.
5451  local function indirect_image(label,sps,tag)
5452    return writer.defer_call(function()
5453      local r,fallback = lookup_reference(label,sps,tag)
5454      if r then
5455        return writer.image(writer.string(label), r.url, r.title)
5456      else
5457        return {"!", fallback}
5458      end
5459    end)
5460  end
```

3.1.5.7 Inline Elements (local)

```
5461  parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5462                     / writer.string
5463
5464  parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
5465                     / writer.string
5466
5467  parsers.Ellipsis = P("...") / writer.ellipsis
5468
5469  parsers.Smart    = parsers.Ellipsis
5470
5471  parsers.Code     = parsers.inticks / writer.code
5472
5473  if options.blankBeforeBlockquote then
5474    parsers.bqstart = parsers.fail
5475  else
5476    parsers.bqstart = parsers.more
5477  end
5478
5479  if options.blankBeforeHeading then
```

```
  parsers.headerstart = parsers.fail
else
  parsers.headerstart = parsers.hash
                      + (parsers.line * (parsers.equal^1 + parsers.dash^1)
                      * parsers.optionalspace * parsers.newline)
end

parsers.EndlineExceptions
                    = parsers.blankline -- paragraph break
                    + parsers.tightblocksep  -- nested list
                    + parsers.eof       -- end of document
                    + parsers.bqstart
                    + parsers.headerstart

parsers.Endline   = parsers.newline
                  * -V("EndlineExceptions")
                  * parsers.spacechar^0
                  / (options.hardLineBreaks and writer.linebreak
                                            or writer.space)

parsers.OptionalIndent
                    = parsers.spacechar^1 / writer.space

parsers.Space      = parsers.spacechar^2 * parsers.Endline / writer.linebreak
                   + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                   + parsers.spacechar^1 * parsers.Endline
                                         * parsers.optionalspace
                                         / (options.hardLineBreaks
                                            and writer.linebreak
                                             or writer.space)
                   + parsers.spacechar^1 * parsers.optionalspace
                                         / writer.space

parsers.NonbreakingEndline
                  = parsers.newline
                  * -V("EndlineExceptions")
                  * parsers.spacechar^0
                  / (options.hardLineBreaks and writer.linebreak
                                            or writer.nbsp)

parsers.NonbreakingSpace
                = parsers.spacechar^2 * parsers.Endline / writer.linebreak
                + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                + parsers.spacechar^1 * parsers.Endline
                                      * parsers.optionalspace
                                      / (options.hardLineBreaks
                                         and writer.linebreak
```

```
                                        or writer.nbsp)
                 + parsers.spacechar^1 * parsers.optionalspace
                                       / writer.nbsp

if options.underscores then
  parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                     parsers.doubleasterisks)
                   + parsers.between(parsers.Inline, parsers.doubleunderscores,
                                     parsers.doubleunderscores)
                   ) / writer.strong

  parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                     parsers.asterisk)
                   + parsers.between(parsers.Inline, parsers.underscore,
                                     parsers.underscore)
                   ) / writer.emphasis
else
  parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                     parsers.doubleasterisks)
                   ) / writer.strong

  parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                     parsers.asterisk)
                   ) / writer.emphasis
end

parsers.AutoLinkUrl    = parsers.less
                         * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
                         * parsers.more
                         / function(url)
                             return writer.link(writer.escape(url), url)
                           end

parsers.AutoLinkEmail = parsers.less
                        * C((parsers.alphanumeric + S("-._+"))^1
                        * P("@") * parsers.urlchar^1)
                        * parsers.more
                        / function(email)
                            return writer.link(writer.escape(email),
                                               "mailto:"..email)
                          end

parsers.AutoLinkRelativeReference
                         = parsers.less
                         * C(parsers.urlchar^1)
                         * parsers.more
                         / function(url)
```

```
                              return writer.link(writer.escape(url), url)
                            end

  parsers.DirectLink    = (parsers.tag / self.parser_functions.parse_inlines_no_link)
                          * parsers.spnl
                          * parsers.lparent
                          * (parsers.url + Cc(""))  -- link can be empty [foo]()
                          * parsers.optionaltitle
                          * parsers.rparent
                          / writer.link

  parsers.IndirectLink  = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
1
                          / indirect_link

  -- parse a link or image (direct or indirect)
  parsers.Link          = parsers.DirectLink + parsers.IndirectLink

  parsers.DirectImage   = parsers.exclamation
                          * (parsers.tag / self.parser_functions.parse_inlines)
                          * parsers.spnl
                          * parsers.lparent
                          * (parsers.url + Cc(""))  -- link can be empty [foo]()
                          * parsers.optionaltitle
                          * parsers.rparent
                          / writer.image

  parsers.IndirectImage = parsers.exclamation * parsers.tag
                          * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image

  parsers.Image         = parsers.DirectImage + parsers.IndirectImage

  -- avoid parsing long strings of * or _ as emph/strong
  parsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
                          / writer.string

  parsers.EscapedChar   = parsers.backslash * C(parsers.escapable) / writer.string

  parsers.InlineHtml    = parsers.emptyelt_any / writer.inline_html_tag
                          + (parsers.htmlcomment / self.parser_functions.parse_inlines_no_
                          / writer.inline_html_comment
                          + parsers.htmlinstruction
                          + parsers.openelt_any / writer.inline_html_tag
                          + parsers.closeelt_any / writer.inline_html_tag

  parsers.HtmlEntity    = parsers.hexentity / entities.hex_entity  / writer.string
                          + parsers.decentity / entities.dec_entity  / writer.string
```

```
                        + parsers.tagentity / entities.char_entity / writer.string
```

3.1.5.8 Block Elements (local)

```
  parsers.DisplayHtml  = (parsers.htmlcomment / self.parser_functions.parse_blocks_neste
                       / writer.block_html_comment
                       + parsers.emptyelt_block / writer.block_html_element
                       + parsers.openelt_exact("hr") / writer.block_html_element
                       + parsers.in_matched_block_tags / writer.block_html_element
                       + parsers.htmlinstruction

  parsers.Verbatim     = Cs( (parsers.blanklines
                           * ((parsers.indentedline - parsers.blankline))^1)^1
                           ) / self.expandtabs / writer.verbatim

  parsers.Blockquote   = Cs(parsers.blockquote_body^1)
                       / self.parser_functions.parse_blocks_nested
                       / writer.blockquote

  parsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
                           + parsers.lineof(parsers.dash)
                           + parsers.lineof(parsers.underscore)
                           ) / writer.hrule

  parsers.Reference    = parsers.define_reference_parser / register_link

  parsers.Paragraph    = parsers.nonindentspace * Ct(parsers.Inline^1)
                       * ( parsers.newline
                       * ( parsers.blankline^1
                         + #parsers.hash
                         + #(parsers.leader * parsers.more * parsers.space^-
1)
                         + parsers.eof
                         )
                       + parsers.eof )
                       / writer.paragraph

  parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
                       / writer.plain
```

3.1.5.9 Lists (local)

```
  parsers.starter = parsers.bullet + parsers.enumerator

  if options.taskLists then
    parsers.tickbox = ( parsers.ticked_box
                      + parsers.halfticked_box
                      + parsers.unticked_box
```

```
                     ) / writer.tickbox
  else
     parsers.tickbox = parsers.fail
  end

  -- we use \001 as a separator between a tight list item and a
  -- nested list under it.
  parsers.NestedList            = Cs((parsers.optionallyindentedline
                                      - parsers.starter)^1)
                                / function(a) return "\001"..a end

  parsers.ListBlockLine         = parsers.optionallyindentedline
                                - parsers.blankline - (parsers.indent^-1
                                                     * parsers.starter)

  parsers.ListBlock             = parsers.line * parsers.ListBlockLine^0

  parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
                                * parsers.ListBlock

  parsers.TightListItem = function(starter)
      return -parsers.HorizontalRule
             * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Neste
1)
               / self.parser_functions.parse_blocks_nested)
             * -(parsers.blanklines * parsers.indent)
  end

  parsers.LooseListItem = function(starter)
      return -parsers.HorizontalRule
             * Cs( starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
               * (parsers.NestedList + parsers.ListContinuationBlock^0)
               * (parsers.blanklines / "\n\n")
               ) / self.parser_functions.parse_blocks_nested
  end

  parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
                       * parsers.skipblanklines * -parsers.bullet
                       + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
                       * parsers.skipblanklines )
                     / writer.bulletlist

  local function ordered_list(items,tight,startnum)
    if options.startNumber then
      startnum = tonumber(startnum) or 1  -- fallback for '#'
      if startnum ~= nil then
        startnum = math.floor(startnum)
```

```
      end
    else
      startnum = nil
    end
    return writer.orderedlist(items,tight,startnum)
  end

  parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
                        ( Ct(parsers.TightListItem(Cb("listtype"))
                            * parsers.TightListItem(parsers.enumerator)^0)
                        * Cc(true) * parsers.skipblanklines * -parsers.enumerator
                        + Ct(parsers.LooseListItem(Cb("listtype"))
                            * parsers.LooseListItem(parsers.enumerator)^0)
                        * Cc(false) * parsers.skipblanklines
                        ) * Cb("listtype") / ordered_list
```

3.1.5.10 Blank (local)

```
  parsers.Blank        = parsers.blankline / ""
                       + parsers.Reference
                       + (parsers.tightblocksep / "\n")
```

3.1.5.11 Headings (local)

```
  -- parse atx header
  parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
                     * parsers.optionalspace
                     * (C(parsers.line)
                       / strip_atx_end
                       / self.parser_functions.parse_inlines)
                     * Cb("level")
                     / writer.heading

  parsers.SetextHeading = #(parsers.line * S("=-"))
                        * Ct(parsers.linechar^1
                            / self.parser_functions.parse_inlines)
                        * parsers.newline
                        * parsers.HeadingLevel
                        * parsers.optionalspace
                        * parsers.newline
                        / writer.heading

  parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
```

3.1.5.12 Syntax Specification Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions`

and returns a conversion function that takes a markdown string and turns it into a plain TEX output.

```
function self.finalize_grammar(extensions)
```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```
  local walkable_syntax = (function(global_walkable_syntax)
    local local_walkable_syntax = {}
    for lhs, rule in pairs(global_walkable_syntax) do
      local_walkable_syntax[lhs] = util.table_copy(rule)
    end
    return local_walkable_syntax
  end)(walkable_syntax)
```

Define `reader->insert_pattern` as a function that receives two arguments: a selector string in the form "⟨*left-hand side terminal symbol*⟩ ⟨*before*, *after*, *or instead of*⟩ ⟨*right-hand side terminal symbol*⟩" and a PEG pattarn to insert. The function adds the pattern to `walkable_syntax`[*left-hand side terminal symbol*] before or after *right-hand side terminal symbol*.

```
  self.insert_pattern = function(selector, pattern)
    local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
    assert(lhs ~= nil,
      [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
      .. selector .. [["]])
    assert(walkable_syntax[lhs] ~= nil,
      [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
    assert(pos == "before" or pos == "after" or pos == "instead of",
      [[Expected positional specifier "before", "after", or "instead of", not "]]
      .. pos .. [["]])
    local rule = walkable_syntax[lhs]
    local index = nil
    for current_index, current_rhs in ipairs(rule) do
      if type(current_rhs) == "string" and current_rhs == rhs then
        index = current_index
        if pos == "after" then
          index = index + 1
        end
        break
      end
    end
    assert(index ~= nil,
      [[Rule ]] .. lhs .. [[ -> ]] .. rhs
        .. [[ does not exist in markdown grammar]])
    if pos == "instead of" then
```

```
5777          rule[index] = pattern
5778        else
5779          table.insert(rule, index, pattern)
5780        end
5781      end
```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```
5782      local syntax =
5783        { "Blocks",
5784
5785          Blocks                = ( V("ExpectedJekyllData")
5786                                  * (V("Blank")^0 / writer.interblocksep)
5787                                  )^-1
5788                                * V("Blank")^0
5789                                * V("Block")^-1
5790                                * (V("Blank")^0 / writer.interblocksep
5791                                  * V("Block"))^0
5792                                * V("Blank")^0 * parsers.eof,
5793
5794          ExpectedJekyllData    = parsers.fail,
5795
5796          Blank                 = parsers.Blank,
5797
5798          Blockquote            = parsers.Blockquote,
5799          Verbatim              = parsers.Verbatim,
5800          HorizontalRule        = parsers.HorizontalRule,
5801          BulletList            = parsers.BulletList,
5802          OrderedList           = parsers.OrderedList,
5803          Heading               = parsers.Heading,
5804          DisplayHtml           = parsers.DisplayHtml,
5805          Paragraph             = parsers.Paragraph,
5806          Plain                 = parsers.Plain,
5807          EndlineExceptions     = parsers.EndlineExceptions,
5808
5809          Str                   = parsers.Str,
5810          Space                 = parsers.Space,
5811          OptionalIndent        = parsers.OptionalIndent,
5812          Endline               = parsers.Endline,
5813          UlOrStarLine          = parsers.UlOrStarLine,
5814          Strong                = parsers.Strong,
5815          Emph                  = parsers.Emph,
5816          Link                  = parsers.Link,
5817          Image                 = parsers.Image,
5818          Code                  = parsers.Code,
5819          AutoLinkUrl           = parsers.AutoLinkUrl,
5820          AutoLinkEmail         = parsers.AutoLinkEmail,
```

```
      AutoLinkRelativeReference
                               = parsers.AutoLinkRelativeReference,
      InlineHtml               = parsers.InlineHtml,
      HtmlEntity               = parsers.HtmlEntity,
      EscapedChar              = parsers.EscapedChar,
      Smart                    = parsers.Smart,
      Symbol                   = parsers.Symbol,
      SpecialChar              = parsers.fail,
    }
```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a PEG pattarn. The function (re)defines `walkable_syntax[`left-hand side terminal symbol`]` to be equal to pattern.

```
  self.update_rule = function(rule_name, pattern)
    assert(syntax[rule_name] ~= nil,
      [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
    walkable_syntax[rule_name] = { pattern }
  end
```

Define a hash table of all characters with special meaning and add method `reader->add_special_character` that extends the hash table and updates the PEG grammar of markdown.

```
  local special_characters = {}
  self.add_special_character = function(c)
    table.insert(special_characters, c)
    syntax.SpecialChar = S(table.concat(special_characters, ""))
  end

  self.add_special_character("*")
  self.add_special_character("`")
  self.add_special_character("[")
  self.add_special_character("]")
  self.add_special_character("<")
  self.add_special_character("!")
  self.add_special_character("\\")
```

Apply syntax extensions.

```
  for _, extension in ipairs(extensions) do
    extension.extend_writer(writer)
    extension.extend_reader(self)
  end
```

Duplicate the `Inline` rule as `IndentedInline` with the right-hand-side terminal symbol `Space` replaced with `OptionalIndent`.

```
  walkable_syntax["IndentedInline"] = util.table_copy(
    walkable_syntax["Inline"])
  self.insert_pattern(
    "IndentedInline instead of Space",
```

```
5856        "OptionalIndent")
```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```
5857    for lhs, rule in pairs(walkable_syntax) do
5858      syntax[lhs] = parsers.fail
5859      for _, rhs in ipairs(rule) do
5860        local pattern
5861        if type(rhs) == "string" then
5862          pattern = V(rhs)
5863        else
5864          pattern = rhs
5865        end
5866        syntax[lhs] = syntax[lhs] + pattern
5867      end
5868    end
```

Finalize the parser by enabling built-in syntax extensions and producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```
5869    if options.underscores then
5870      self.add_special_character("_")
5871    end
5872
5873    if not options.codeSpans then
5874      syntax.Code = parsers.fail
5875    end
5876
5877    if not options.html then
5878      syntax.DisplayHtml = parsers.fail
5879      syntax.InlineHtml = parsers.fail
5880      syntax.HtmlEntity  = parsers.fail
5881    else
5882      self.add_special_character("&")
5883    end
5884
5885    if options.preserveTabs then
5886      options.stripIndent = false
5887    end
5888
5889    if not options.smartEllipses then
5890      syntax.Smart = parsers.fail
5891    else
5892      self.add_special_character(".")
5893    end
5894
```

```
    if not options.relativeReferences then
      syntax.AutoLinkRelativeReference = parsers.fail
    end

    local blocks_nested_t = util.table_copy(syntax)
    blocks_nested_t.ExpectedJekyllData = parsers.fail
    parsers.blocks_nested = Ct(blocks_nested_t)

    parsers.blocks = Ct(syntax)

    local inlines_t = util.table_copy(syntax)
    inlines_t[1] = "Inlines"
    inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
    parsers.inlines = Ct(inlines_t)

    local inlines_no_link_t = util.table_copy(inlines_t)
    inlines_no_link_t.Link = parsers.fail
    parsers.inlines_no_link = Ct(inlines_no_link_t)

    local inlines_no_inline_note_t = util.table_copy(inlines_t)
    inlines_no_inline_note_t.InlineNote = parsers.fail
    parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)

    local inlines_no_html_t = util.table_copy(inlines_t)
    inlines_no_html_t.DisplayHtml = parsers.fail
    inlines_no_html_t.InlineHtml = parsers.fail
    inlines_no_html_t.HtmlEntity = parsers.fail
    parsers.inlines_no_html = Ct(inlines_no_html_t)

    local inlines_nbsp_t = util.table_copy(inlines_t)
    inlines_nbsp_t.Endline = parsers.NonbreakingEndline
    inlines_nbsp_t.Space = parsers.NonbreakingSpace
    parsers.inlines_nbsp = Ct(inlines_nbsp_t)
```

Return a function that converts markdown string `input` into a plain TEX output and returns it. Note that the converter assumes that the input has UNIX line endings.

```
    return function(input)
      references = {}
```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```
      local opt_string = {}
      for k, _ in pairs(defaultOptions) do
        local v = options[k]
        if type(v) == "table" then
          for _, i in ipairs(v) do
            opt_string[#opt_string+1] = k .. "=" .. tostring(i)
```

```
5936            end
5937          elseif k ~= "cacheDir" then
5938            opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5939          end
5940        end
5941        table.sort(opt_string)
5942        local salt = table.concat(opt_string, ",") .. "," .. metadata.version
5943        local output
```

If we cache markdown documents, produce the cache file and transform its filename to plain TEX output via the `writer->pack` method.

```
5944        local function convert(input)
5945          local document = self.parser_functions.parse_blocks(input)
5946          return util.rope_to_string(writer.document(document))
5947        end
5948        if options.eagerCache or options.finalizeCache then
5949          local name = util.cache(options.cacheDir, input, salt, convert,
5950                                  ".md" .. writer.suffix)
5951          output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
5952        else
5953          output = convert(input)
5954        end
```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```
5955        if options.finalizeCache then
5956          local file, mode
5957          if options.frozenCacheCounter > 0 then
5958            mode = "a"
5959          else
5960            mode = "w"
5961          end
5962          file = assert(io.open(options.frozenCacheFileName, mode),
5963            [[Could not open file "]] .. options.frozenCacheFileName
5964            .. [[" for writing]])
5965          assert(file:write([[\expandafter\global\expandafter\def\csname ]]
5966            .. [[markdownFrozenCache]] .. options.frozenCacheCounter
5967            .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
5968          assert(file:close())
5969        end
5970        return output
5971      end
5972    end
5973    return self
5974  end
```

3.1.6 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```
5975 M.extensions = {}
```

3.1.6.1 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```
5976 M.extensions.citations = function(citation_nbsps)
```

Define table `escaped_citation_chars` containing the characters to escape in citations.

```
5977   local escaped_citation_chars = {
5978     ["{"] = "\\markdownRendererLeftBrace{}",
5979     ["}"] = "\\markdownRendererRightBrace{}",
5980     ["%"] = "\\markdownRendererPercentSign{}",
5981     ["\\"] = "\\markdownRendererBackslash{}",
5982     ["#"] = "\\markdownRendererHash{}",
5983   }
5984   return {
5985     extend_writer = function(self)
5986       local options = self.options
5987
```

Use the `escaped_citation_chars` to create the `escape_citation` escaper functions.

```
5988       local escape_citation = util.escaper(
5989         escaped_citation_chars,
5990         self.escaped_minimal_strings)
```

Define `writer->citation` as a function that will transform an input citation name `c` to the output format. If `writer->hybrid` is `true`, use the `writer->escape_minimal` function. Otherwise, use the `escape_citation` function.

```
5991       if options.hybrid then
5992         self.citation = self.escape_minimal
5993       else
5994         self.citation = escape_citation
5995       end
```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, the citations should

be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```
5996      function self.citations(text_cites, cites)
5997        local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
5998          "{", #cites, "}"}
5999        for _,cite in ipairs(cites) do
6000          buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6001            cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"}
6002        end
6003        return buffer
6004      end
6005    end, extend_reader = function(self)
6006      local parsers = self.parsers
6007      local writer = self.writer
6008
6009      local citation_chars
6010                    = parsers.alphanumeric
6011                    + S("#$%&-+<>~/_")
6012
6013      local citation_name
6014                    = Cs(parsers.dash^-1) * parsers.at
6015                    * Cs(citation_chars
6016                        * (((citation_chars + parsers.internal_punctuation
6017                            - parsers.comma - parsers.semicolon)
6018                           * -#((parsers.internal_punctuation - parsers.comma
6019                                - parsers.semicolon)^0
6020                               * -(citation_chars + parsers.internal_punctuation
6021                                  - parsers.comma - parsers.semicolon)))^0
6022                          * citation_chars)^-1)
6023
6024      local citation_body_prenote
6025                    = Cs((parsers.alphanumeric^1
6026                         + parsers.bracketed
6027                         + parsers.inticks
6028                         + (parsers.anyescaped
6029                           - (parsers.rbracket + parsers.blankline^2))
```

```
                        - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)

    local citation_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + (parsers.anyescaped
                         - (parsers.rbracket + parsers.semicolon
                           + parsers.blankline^2))
                       - (parsers.spnl * parsers.rbracket))^0)

    local citation_body_chunk
                  = citation_body_prenote
                  * parsers.spnl * citation_name
                  * (parsers.internal_punctuation - parsers.semicolon)^-
1
                  * parsers.spnl * citation_body_postnote

    local citation_body
                  = citation_body_chunk
                  * (parsers.semicolon * parsers.spnl
                    * citation_body_chunk)^0

    local citation_headless_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + (parsers.anyescaped
                         - (parsers.rbracket + parsers.at
                           + parsers.semicolon + parsers.blankline^2))
                       - (parsers.spnl * parsers.rbracket))^0)

    local citation_headless_body
                  = citation_headless_body_postnote
                  * (parsers.sp * parsers.semicolon * parsers.spnl
                    * citation_body_chunk)^0

    local citations
                  = function(text_cites, raw_cites)
        local function normalize(str)
            if str == "" then
                str = nil
            else
                str = (citation_nbsps and
                  self.parser_functions.parse_inlines_nbsp or
                  self.parser_functions.parse_inlines)(str)
            end
```

```
6076                return str
6077            end
6078
6079            local cites = {}
6080            for i = 1,#raw_cites,4 do
6081                cites[#cites+1] = {
6082                    prenote = normalize(raw_cites[i]),
6083                    suppress_author = raw_cites[i+1] == "-",
6084                    name = writer.citation(raw_cites[i+2]),
6085                    postnote = normalize(raw_cites[i+3]),
6086                }
6087            end
6088            return writer.citations(text_cites, cites)
6089        end
6090
6091        local TextCitations
6092                      = Ct((parsers.spnl
6093                      * Cc("")
6094                      * citation_name
6095                      * ((parsers.spnl
6096                          * parsers.lbracket
6097                          * citation_headless_body
6098                          * parsers.rbracket) + Cc("")))^1)
6099                      / function(raw_cites)
6100                          return citations(true, raw_cites)
6101                        end
6102
6103        local ParenthesizedCitations
6104                      = Ct((parsers.spnl
6105                      * parsers.lbracket
6106                      * citation_body
6107                      * parsers.rbracket)^1)
6108                      / function(raw_cites)
6109                          return citations(false, raw_cites)
6110                        end
6111
6112        local Citations = TextCitations + ParenthesizedCitations
6113
6114        self.insert_pattern("Inline after Emph", Citations)
6115
6116        self.add_special_character("@")
6117        self.add_special_character("-")
6118      end
6119    }
6120  end
```

3.1.6.2 Content Blocks

The `extensions.content_blocks` function implements

the iA,Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
M.extensions.content_blocks = function(language_map)
```

The `languages_json` table maps programming language filename extensions to fence infostrings. All `language_map` files located by the kpathsea library are loaded into a chain of tables. `languages_json` corresponds to the first table and is chained with the rest via Lua metatables.

```
  local languages_json = (function()
    local base, prev, curr
    for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
      local file = io.open(pathname, "r")
      if not file then goto continue end
      local input = assert(file:read("*a"))
      assert(file:close())
      local json = input:gsub('("[^\n]-"):','[%1]=')
      curr = load("_ENV = {}; return "..json)()
      if type(curr) == "table" then
        if base == nil then
          base = curr
        else
          setmetatable(prev, { __index = curr })
        end
        prev = curr
      end
      ::continue::
    end
    return base or {}
  end)()

  return {
    extend_writer = function(self)
```

Define `writer->contentblock` as a function that will transform an input iA,Writer content block to the output format, where `src` corresponds to the URI prefix, `suf` to the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```
      function self.contentblock(src,suf,type,tit)
        if not self.is_writing then return "" end
        src = src.."."..suf
        suf = suf:lower()
        if type == "onlineimage" then
          return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
                                  "{",self.string(src),"}",
                                  "{",self.uri(src),"}",
                                  "{",self.string(tit or ""),"}"}
```

```
      elseif languages_json[suf] then
        return {"\\markdownRendererContentBlockCode{",suf,"}",
                                "{",self.string(languages_json[suf]),"}",
                                "{",self.string(src),"}",
                                "{",self.uri(src),"}",
                                "{",self.string(tit or ""),"}"}
      else
        return {"\\markdownRendererContentBlock{",suf,"}",
                                "{",self.string(src),"}",
                                "{",self.uri(src),"}",
                                "{",self.string(tit or ""),"}"}
      end
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local contentblock_tail
                 = parsers.optionaltitle
                 * (parsers.newline + parsers.eof)

    -- case insensitive online image suffix:
    local onlineimagesuffix
                 = (function(...)
                     local parser = nil
                     for _, suffix in ipairs({...}) do
                       local pattern=nil
                       for i=1,#suffix do
                         local char=suffix:sub(i,i)
                         char = S(char:lower()..char:upper())
                         if pattern == nil then
                           pattern = char
                         else
                           pattern = pattern * char
                         end
                       end
                       if parser == nil then
                         parser = pattern
                       else
                         parser = parser + pattern
                       end
                     end
                     return parser
                   end)("png", "jpg", "jpeg", "gif", "tif", "tiff")

    -- online image url for iA Writer content blocks with mandatory suffix,
    -- allowing nested brackets:
```

```
      local onlineimageurl
                    = (parsers.less
                      * Cs((parsers.anyescaped
                           - parsers.more
                           - #(parsers.period
                              * onlineimagesuffix
                              * parsers.more
                              * contentblock_tail))^0)
                      * parsers.period
                      * Cs(onlineimagesuffix)
                      * parsers.more
                      + (Cs((parsers.inparens
                            + (parsers.anyescaped
                              - parsers.spacing
                              - parsers.rparent
                              - #(parsers.period
                                 * onlineimagesuffix
                                 * contentblock_tail)))^0)
                        * parsers.period
                        * Cs(onlineimagesuffix))
                      ) * Cc("onlineimage")

      -- filename for iA Writer content blocks with mandatory suffix:
      local localfilepath
                    = parsers.slash
                    * Cs((parsers.anyescaped
                         - parsers.tab
                         - parsers.newline
                         - #(parsers.period
                            * parsers.alphanumeric^1
                            * contentblock_tail))^1)
                    * parsers.period
                    * Cs(parsers.alphanumeric^1)
                    * Cc("localfile")

      local ContentBlock
                    = parsers.leader
                    * (localfilepath + onlineimageurl)
                    * contentblock_tail
                    / writer.contentblock

      self.insert_pattern("Block before Blockquote", ContentBlock)
    end
  }
end
```

3.1.6.3 Definition Lists

The `extensions.definition_lists` function implements

the definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```
M.extensions.definition_lists = function(tight_lists)
  return {
    extend_writer = function(self)
```

Define `writer->definitionlist` as a function that will transform an input definition list to the output format, where `items` is an array of tables, each of the form `{ term = t, definitions = defs }`, where `t` is a term and `defs` is an array of definitions. `tight` specifies, whether the list is tight or not.

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
                               "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

      local defstart = ( defstartchar * #parsers.spacing
                                      * (parsers.tab + parsers.space^-3)
                       + parsers.space * defstartchar * #parsers.spacing
                                       * (parsers.tab + parsers.space^-2)
                       + parsers.space * parsers.space * defstartchar
                                       * #parsers.spacing
                                       * (parsers.tab + parsers.space^-1)
```

```
                      + parsers.space * parsers.space * parsers.space
                                      * defstartchar * #parsers.spacing
                      )

      local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

      local function definition_list_item(term, defs, _)
        return { term = self.parser_functions.parse_inlines(term),
                 definitions = defs }
      end

      local DefinitionListItemLoose
                    = C(parsers.line) * parsers.skipblanklines
                    * Ct((defstart
                         * parsers.indented_blocks(dlchunk)
                         / self.parser_functions.parse_blocks_nested)^1)
                    * Cc(false) / definition_list_item

      local DefinitionListItemTight
                    = C(parsers.line)
                    * Ct((defstart * dlchunk
                         / self.parser_functions.parse_blocks_nested)^1)
                    * Cc(true) / definition_list_item

      local DefinitionList
                    = ( Ct(DefinitionListItemLoose^1) * Cc(false)
                      + Ct(DefinitionListItemTight^1)
                      * (parsers.skipblanklines
                        * -DefinitionListItemLoose * Cc(true))
                      ) / writer.definitionlist

      self.insert_pattern("Block after Heading", DefinitionList)
    end
  }
end
```

3.1.6.4 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires between a paragraph and the following fenced code block.

```
M.extensions.fenced_code = function(blank_before_code_fence)
  return {
    extend_writer = function(self)
      local options = self.options

```

Define `writer->codeFence` as a function that will transform an input fenced code block `s` with the infostring `i` to the output format.

```
6327      function self.fencedCode(i, s)
6328        if not self.is_writing then return "" end
6329        s = string.gsub(s, '[\r\n%s]*$', '')
6330        local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6331        return {"\\markdownRendererInputFencedCode{",name,"}{",i,"}"}
6332      end
6333    end, extend_reader = function(self)
6334      local parsers = self.parsers
6335      local writer = self.writer
6336
6337      local function captures_geq_length(_,i,a,b)
6338        return #a >= #b and i
6339      end
6340
6341      local infostring   = (parsers.linechar - (parsers.backtick
6342                          + parsers.space^1 * (parsers.newline + parsers.eof)))^0
6343
6344      local fenceindent
6345      local fencehead    = function(char)
6346        return             C(parsers.nonindentspace) / function(s) fenceindent = #s end
6347                           * Cg(char^3, "fencelength")
6348                           * parsers.optionalspace * C(infostring)
6349                           * parsers.optionalspace * (parsers.newline + parsers.eof)
6350      end
6351
6352      local fencetail    = function(char)
6353        return             parsers.nonindentspace
6354                           * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6355                           * parsers.optionalspace * (parsers.newline + parsers.eof)
6356                           + parsers.eof
6357      end
6358
6359      local fencedline   = function(char)
6360        return             C(parsers.line - fencetail(char))
6361                           / function(s)
6362                               local i = 1
6363                               local remaining = fenceindent
6364                               while true do
6365                                 local c = s:sub(i, i)
6366                                 if c == " " and remaining > 0 then
6367                                   remaining = remaining - 1
6368                                   i = i + 1
6369                                 elseif c == "\t" and remaining > 3 then
6370                                   remaining = remaining - 4
6371                                   i = i + 1
```

```
6372                                else
6373                                  break
6374                                end
6375                              end
6376                              return s:sub(i)
6377                            end
6378       end
6379
6380       local TildeFencedCode
6381              = fencehead(parsers.tilde)
6382              * Cs(fencedline(parsers.tilde)^0)
6383              * fencetail(parsers.tilde)
6384
6385       local BacktickFencedCode
6386              = fencehead(parsers.backtick)
6387              * Cs(fencedline(parsers.backtick)^0)
6388              * fencetail(parsers.backtick)
6389
6390       local FencedCode = (TildeFencedCode
6391                             + BacktickFencedCode)
6392                           / function(infostring, code)
6393                               return writer.fencedCode(writer.string(infostring),
6394                                                        self.expandtabs(code))
6395                             end
6396
6397       self.insert_pattern("Block after Verbatim",
6398                           FencedCode)
6399
6400       local fencestart
6401       if blank_before_code_fence then
6402         fencestart = parsers.fail
6403       else
6404         fencestart = fencehead(parsers.backtick)
6405                    + fencehead(parsers.tilde)
6406       end
6407
6408       local EndlineExceptions = parsers.EndlineExceptions + fencestart
6409       self.update_rule("EndlineExceptions", EndlineExceptions)
6410
6411       self.add_special_character("~")
6412     end
6413   }
6414 end
```

3.1.6.5 Footnotes The `extensions.footnotes` function implements the Pandoc footnote and inline footnote syntax extensions. When the `footnote` parame-

ter is `true`, the Pandoc footnote syntax extension will be enabled. When the `inline_footnotes` parameter is `true`, the Pandoc inline footnote syntax extension will be enabled.

```
M.extensions.footnotes = function(footnotes, inline_footnotes)
  assert(footnotes or inline_footnotes)
  return {
    extend_writer = function(self)
```

Define `writer->note` as a function that will transform an input footnote `s` to the output format.

```
      function self.note(s)
        return {"\\markdownRendererFootnote{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      if inline_footnotes then
        local InlineNote
                    = parsers.circumflex
                    * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
                    / writer.note

        self.insert_pattern("Inline after Emph", InlineNote)
      end
      if footnotes then
        local function strip_first_char(s)
          return s:sub(2)
        end

        local RawNoteRef
                      = #(parsers.lbracket * parsers.circumflex)
                      * parsers.tag / strip_first_char

        local rawnotes = {}

        -- like indirect_link
        local function lookup_note(ref)
          return writer.defer_call(function()
            local found = rawnotes[self.normalize_tag(ref)]
            if found then
              return writer.note(
                self.parser_functions.parse_blocks_nested(found))
            else
              return {"[",
                self.parser_functions.parse_inlines("^" .. ref), "]"}
            end
```

```
6456          end)
6457        end
6458
6459        local function register_note(ref,rawnote)
6460          rawnotes[self.normalize_tag(ref)] = rawnote
6461          return ""
6462        end
6463
6464        local NoteRef = RawNoteRef / lookup_note
6465
6466        local NoteBlock
6467                    = parsers.leader * RawNoteRef * parsers.colon
6468                    * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6469                    / register_note
6470
6471        local Blank = NoteBlock + parsers.Blank
6472        self.update_rule("Blank", Blank)
6473
6474        self.insert_pattern("Inline after Emph", NoteRef)
6475      end
6476
6477      self.add_special_character("^")
6478    end
6479  }
6480 end
```

3.1.6.6 Header Attributes

The `extensions.header_attributes` function implements a syntax extension that enables the assignment of HTML attributes to headings.

```
6481 M.extensions.header_attributes = function()
6482   return {
6483     extend_writer = function()
6484     end, extend_reader = function(self)
6485       local parsers = self.parsers
6486       local writer = self.writer
6487
6488       parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
6489                          * parsers.optionalspace
6490                          * (C(((parsers.linechar
6491                                - ((parsers.hash^1
6492                                   * parsers.optionalspace
6493                                   * parsers.HeadingAttributes^-1
6494                                   + parsers.HeadingAttributes)
6495                                  * parsers.optionalspace
6496                                  * parsers.newline))
6497                               * (parsers.linechar
6498                                 - parsers.hash
```

```
                                  - parsers.lbrace)^0)^1)
                               / self.parser_functions.parse_inlines)
                           * Cg(Ct(parsers.newline
                                  + (parsers.hash^1
                                    * parsers.optionalspace
                                    * parsers.HeadingAttributes^-1
                                    + parsers.HeadingAttributes)
                                  * parsers.optionalspace
                                  * parsers.newline), "attributes")
                           * Cb("level")
                           * Cb("attributes")
                           / writer.heading

      parsers.SetextHeading = #(parsers.line * S("=-"))
                              * (C(((parsers.linechar
                                     - (parsers.HeadingAttributes
                                       * parsers.optionalspace
                                       * parsers.newline))
                                    * (parsers.linechar
                                      - parsers.lbrace)^0)^1)
                                 / self.parser_functions.parse_inlines)
                              * Cg(Ct(parsers.newline
                                     + (parsers.HeadingAttributes
                                       * parsers.optionalspace
                                       * parsers.newline)), "attributes")
                              * parsers.HeadingLevel
                              * Cb("attributes")
                              * parsers.optionalspace
                              * parsers.newline
                              / writer.heading

      local Heading = parsers.AtxHeading + parsers.SetextHeading
      self.update_rule("Heading", Heading)
    end
  }
end
```

3.1.6.7 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata

```
M.extensions.jekyll_data = function(expect_jekyll_data)
  return {
    extend_writer = function(self)
```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p`

is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```
6538        function self.jekyllData(d, t, p)
6539          if not self.is_writing then return "" end
6540
6541          local buf = {}
6542
6543          local keys = {}
6544          for k, _ in pairs(d) do
6545            table.insert(keys, k)
6546          end
6547          table.sort(keys)
6548
6549          if not p then
6550            table.insert(buf, "\\markdownRendererJekyllDataBegin")
6551          end
6552
6553          if #d > 0 then
6554              table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
6555              table.insert(buf, self.uri(p or "null"))
6556              table.insert(buf, "}{")
6557              table.insert(buf, #keys)
6558              table.insert(buf, "}")
6559          else
6560              table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
6561              table.insert(buf, self.uri(p or "null"))
6562              table.insert(buf, "}{")
6563              table.insert(buf, #keys)
6564              table.insert(buf, "}")
6565          end
6566
6567          for _, k in ipairs(keys) do
6568            local v = d[k]
6569            local typ = type(v)
6570            k = tostring(k or "null")
6571            if typ == "table" and next(v) ~= nil then
6572              table.insert(
6573                buf,
6574                self.jekyllData(v, t, k)
6575              )
6576            else
6577              k = self.uri(k)
6578              v = tostring(v)
6579              if typ == "boolean" then
6580                table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6581                table.insert(buf, k)
```

```
6582              table.insert(buf, "}{")
6583              table.insert(buf, v)
6584              table.insert(buf, "}")
6585            elseif typ == "number" then
6586              table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6587              table.insert(buf, k)
6588              table.insert(buf, "}{")
6589              table.insert(buf, v)
6590              table.insert(buf, "}")
6591            elseif typ == "string" then
6592              table.insert(buf, "\\markdownRendererJekyllDataString{")
6593              table.insert(buf, k)
6594              table.insert(buf, "}{")
6595              table.insert(buf, t(v))
6596              table.insert(buf, "}")
6597            elseif typ == "table" then
6598              table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
6599              table.insert(buf, k)
6600              table.insert(buf, "}")
6601            else
6602              error(format("Unexpected type %s for value of " ..
6603                           "YAML key %s", typ, k))
6604            end
6605          end
6606        end
6607
6608        if #d > 0 then
6609          table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6610        else
6611          table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6612        end
6613
6614        if not p then
6615          table.insert(buf, "\\markdownRendererJekyllDataEnd")
6616        end
6617
6618        return buf
6619      end
6620    end, extend_reader = function(self)
6621      local parsers = self.parsers
6622      local writer = self.writer
6623
6624      local JekyllData
6625                  = Cmt( C((parsers.line - P("---") - P("..."))^0)
6626                       , function(s, i, text) -- luacheck: ignore s i
6627                           local data
6628                           local ran_ok, _ = pcall(function()
```

```
                              local tinyyaml = require("markdown-tinyyaml")
                              data = tinyyaml.parse(text, {timestamps=false})
                            end)
                            if ran_ok and data ~= nil then
                              return true, writer.jekyllData(data, function(s)
                                return self.parser_functions.parse_blocks_nested(s)
                              end, nil)
                            else
                              return false
                            end
                          end
                        )

      local UnexpectedJekyllData
                    = P("---")
                    * parsers.blankline / 0
                    * #(-parsers.blankline)  -- if followed by blank, it's an hrule
                    * JekyllData
                    * (P("---") + P("..."))

      local ExpectedJekyllData
                    = ( P("---")
                      * parsers.blankline / 0
                      * #(-parsers.blankline)  -- if followed by blank, it's an hrule
                      )^-1
                    * JekyllData
                    * (P("---") + P("..."))^-1

      self.insert_pattern("Block before Blockquote", UnexpectedJekyllData)
      if expect_jekyll_data then
        self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
      end
    end
  }
end
```

3.1.6.8 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (affectionately known as pipe tables). When the `table_captions` parameter is `true`, the function also implements the Pandoc `table_captions` syntax extension for table captions.

```
M.extensions.pipe_tables = function(table_captions)

  local function make_pipe_table_rectangular(rows)
    local num_columns = #rows[2]
    local rectangular_rows = {}
    for i = 1, #rows do
```

```
6670      local row = rows[i]
6671      local rectangular_row = {}
6672      for j = 1, num_columns do
6673        rectangular_row[j] = row[j] or ""
6674      end
6675      table.insert(rectangular_rows, rectangular_row)
6676    end
6677    return rectangular_rows
6678  end
6679
6680  local function pipe_table_row(allow_empty_first_column
6681                                , nonempty_column
6682                                , column_separator
6683                                , column)
6684    local row_beginning
6685    if allow_empty_first_column then
6686      row_beginning = -- empty first column
6687                      #(parsers.spacechar^4
6688                        * column_separator)
6689                    * parsers.optionalspace
6690                    * column
6691                    * parsers.optionalspace
6692                    -- non-empty first column
6693                    + parsers.nonindentspace
6694                    * nonempty_column^-1
6695                    * parsers.optionalspace
6696    else
6697      row_beginning = parsers.nonindentspace
6698                    * nonempty_column^-1
6699                    * parsers.optionalspace
6700    end
6701
6702    return Ct(row_beginning
6703              * (-- single column with no leading pipes
6704                 #(column_separator
6705                   * parsers.optionalspace
6706                   * parsers.newline)
6707                 * column_separator
6708                 * parsers.optionalspace
6709                 -- single column with leading pipes or
6710                 -- more than a single column
6711                 + (column_separator
6712                   * parsers.optionalspace
6713                   * column
6714                   * parsers.optionalspace)^1
6715                 * (column_separator
6716                   * parsers.optionalspace)^-1))
```

```
  end

  return {
    extend_writer = function(self)
```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```
      function self.table(rows, caption)
        if not self.is_writing then return "" end
        local buffer = {"\\markdownRendererTable{",
          caption or "", "}{", #rows - 1, "}{", #rows[1], "}"}
        local temp = rows[2] -- put alignments on the first row
        rows[2] = rows[1]
        rows[1] = temp
        for i, row in ipairs(rows) do
          table.insert(buffer, "{")
          for _, column in ipairs(row) do
            if i > 1 then -- do not use braces for alignments
              table.insert(buffer, "{")
            end
            table.insert(buffer, column)
            if i > 1 then
              table.insert(buffer, "}")
            end
          end
          table.insert(buffer, "}")
        end
        return buffer
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local table_hline_separator = parsers.pipe + parsers.plus

      local table_hline_column = (parsers.dash
                                  - #(parsers.dash
                                     * (parsers.spacechar
                                       + table_hline_separator
                                       + parsers.newline)))^1
                               * (parsers.colon * Cc("r")
                                 + parsers.dash * Cc("d"))
                               + parsers.colon
                               * (parsers.dash
                                 - #(parsers.dash
                                    * (parsers.spacechar
```

```
                                      + table_hline_separator
                                      + parsers.newline)))^1
                                * (parsers.colon * Cc("c")
                                  + parsers.dash * Cc("l"))

      local table_hline = pipe_table_row(false
                                        , table_hline_column
                                        , table_hline_separator
                                        , table_hline_column)

      local table_caption_beginning = parsers.skipblanklines
                                    * parsers.nonindentspace
                                    * (P("Table")^-1 * parsers.colon)
                                    * parsers.optionalspace

      local table_row = pipe_table_row(true
                                      , (C((parsers.linechar - parsers.pipe)^1)
                                        / self.parser_functions.parse_inlines)
                                      , parsers.pipe
                                      , (C((parsers.linechar - parsers.pipe)^0)
                                        / self.parser_functions.parse_inlines))

      local table_caption
      if table_captions then
        table_caption = #table_caption_beginning
                      * table_caption_beginning
                      * Ct(parsers.IndentedInline^1)
                      * parsers.newline
      else
        table_caption = parsers.fail
      end

      local PipeTable = Ct(table_row * parsers.newline
                        * table_hline
                        * (parsers.newline * table_row)^0)
                      / make_pipe_table_rectangular
                      * table_caption^-1
                      / writer.table

      self.insert_pattern("Block after Blockquote", PipeTable)
    end
  }
end
```

3.1.6.9 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
M.extensions.strike_through = function()
  return {
    extend_writer = function(self)
```

Define `writer->strike_through` as a function that will transform a strike-through span `s` of input text to the output format.

```
      function self.strike_through(s)
        return {"\\markdownRendererStrikeThrough{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local StrikeThrough = (
        parsers.between(parsers.Inline, parsers.doubletildes,
                        parsers.doubletildes)
      ) / writer.strike_through

      self.insert_pattern("Inline after Emph", StrikeThrough)

      self.add_special_character("~")
    end
  }
end
```

3.1.6.10 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```
M.extensions.superscripts = function()
  return {
    extend_writer = function(self)
```

Define `writer->superscript` as a function that will transform a superscript span `s` of input text to the output format.

```
      function self.superscript(s)
        return {"\\markdownRendererSuperscript{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Superscript = (
        parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
      ) / writer.superscript

      self.insert_pattern("Inline after Emph", Superscript)

      self.add_special_character("^")
```

```
    end
  }
end
```

3.1.6.11 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
M.extensions.subscripts = function()
  return {
    extend_writer = function(self)
```

Define `writer->subscript` as a function that will transform a subscript span `s` of input text to the output format.

```
      function self.subscript(s)
        return {"\\markdownRendererSubscript{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Subscript = (
        parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
      ) / writer.subscript

      self.insert_pattern("Inline after Emph", Subscript)

      self.add_special_character("~")
    end
  }
end
```

3.1.6.12 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list extension.

```
M.extensions.fancy_lists = function()
  return {
    extend_writer = function(self)
      local options = self.options

```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,

  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,
  - `OneParen` – parentheses, and
  - `Period` – periods.

```
6869      function self.fancylist(items,tight,startnum,numstyle,numdelim)
6870        if not self.is_writing then return "" end
6871        local buffer = {}
6872        local num = startnum
6873        for _,item in ipairs(items) do
6874          buffer[#buffer + 1] = self.fancyitem(item,num)
6875          if num ~= nil then
6876            num = num + 1
6877          end
6878        end
6879        local contents = util.intersperse(buffer,"\n")
6880        if tight and options.tightLists then
6881          return {"\\markdownRendererFancyOlBeginTight{",
6882                  numstyle,"}{",numdelim,"}",contents,
6883                  "\n\\markdownRendererFancyOlEndTight "}
6884        else
6885          return {"\\markdownRendererFancyOlBegin{",
6886                  numstyle,"}{",numdelim,"}",contents,
6887                  "\n\\markdownRendererFancyOlEnd "}
6888        end
6889      end
```

Define `writer->fancyitem` as a function that will transform an input fancy ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```
6890      function self.fancyitem(s,num)
6891        if num ~= nil then
6892          return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
6893                  "\\markdownRendererFancyOlItemEnd "}
6894        else
6895          return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEnd "
6896        end
6897      end
6898    end, extend_reader = function(self)
6899      local parsers = self.parsers
6900      local options = self.options
6901      local writer = self.writer
6902
```

```
6903        local label = parsers.dig + parsers.letter
6904        local numdelim = parsers.period + parsers.rparent
6905        local enumerator = C(label^3 * numdelim) * #parsers.spacing
6906                         + C(label^2 * numdelim) * #parsers.spacing
6907                                                 * (parsers.tab + parsers.space^1)
6908                         + C(label * numdelim) * #parsers.spacing
6909                                               * (parsers.tab + parsers.space^-2)
6910                         + parsers.space * C(label^2 * numdelim)
6911                                               * #parsers.spacing
6912                         + parsers.space * C(label * numdelim)
6913                                               * #parsers.spacing
6914                                               * (parsers.tab + parsers.space^-1)
6915                         + parsers.space * parsers.space * C(label^1
6916                                               * numdelim) * #parsers.spacing
6917        local starter = parsers.bullet + enumerator
6918
6919        local NestedList = Cs((parsers.optionallyindentedline
6920                             - starter)^1)
6921                         / function(a) return "\001"..a end
6922
6923        local ListBlockLine  = parsers.optionallyindentedline
6924                             - parsers.blankline - (parsers.indent^-1
6925                                                  * starter)
6926
6927        local ListBlock = parsers.line * ListBlockLine^0
6928
6929        local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6930                                    * ListBlock
6931
6932        local TightListItem = function(starter)
6933            return -parsers.HorizontalRule
6934                   * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
    1)
6935                     / self.parser_functions.parse_blocks_nested)
6936                   * -(parsers.blanklines * parsers.indent)
6937        end
6938
6939        local LooseListItem = function(starter)
6940            return -parsers.HorizontalRule
6941                   * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6942                     * (NestedList + ListContinuationBlock^0)
6943                     * (parsers.blanklines / "\n\n")
6944                     ) / self.parser_functions.parse_blocks_nested
6945        end
6946
6947        local function roman2number(roman)
6948          local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
```

```
6949          local numeral = 0
6950
6951          local i = 1
6952          local len = string.len(roman)
6953          while i < len do
6954            local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman, i+
6955            if z1 < z2 then
6956                numeral = numeral + (z2 - z1)
6957                i = i + 2
6958            else
6959                numeral = numeral + z1
6960                i = i + 1
6961            end
6962          end
6963          if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
6964          return numeral
6965        end
6966
6967        local function sniffstyle(itemprefix)
6968          local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.)]*)")
6969          local numdelim
6970          if delimend == ")" then
6971            numdelim = "OneParen"
6972          elseif delimend == "." then
6973            numdelim = "Period"
6974          else
6975            numdelim = "Default"
6976          end
6977          numstr = numstr or itemprefix
6978
6979          local num
6980          num = numstr:match("^([IVXL]+)")
6981          if num then
6982            return roman2number(num), "UpperRoman", numdelim
6983          end
6984          num = numstr:match("^([ivxl]+)")
6985          if num then
6986            return roman2number(string.upper(num)), "LowerRoman", numdelim
6987          end
6988          num = numstr:match("^([A-Z])")
6989          if num then
6990            return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
6991          end
6992          num = numstr:match("^([a-z])")
6993          if num then
6994            return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
6995          end
```

```
6996          return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
6997        end
6998
6999        local function fancylist(items,tight,start)
7000          local startnum, numstyle, numdelim = sniffstyle(start)
7001          return writer.fancylist(items,tight,
7002                                  options.startNumber and startnum,
7003                                  numstyle or "Decimal",
7004                                  numdelim or "Default")
7005        end
7006
7007        local FancyList = Cg(enumerator, "listtype") *
7008                        ( Ct(TightListItem(Cb("listtype"))
7009                            * TightListItem(enumerator)^0)
7010                        * Cc(true) * parsers.skipblanklines * -enumerator
7011                        + Ct(LooseListItem(Cb("listtype"))
7012                            * LooseListItem(enumerator)^0)
7013                        * Cc(false) * parsers.skipblanklines
7014                        ) * Cb("listtype") / fancylist
7015
7016        self.update_rule("OrderedList", FancyList)
7017      end
7018    }
7019  end
```

3.1.7 Conversion from Markdown to Plain TEX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain TEX output. See Section **??**.

```
7020 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
7021   options = options or {}
7022   setmetatable(options, { __index = function (_, key)
7023     return defaultOptions[key] end })
```

Apply built-in syntax extensions based on `options`.

```
7024   local extensions = {}
7025
7026   if options.contentBlocks then
7027     local content_blocks_extension = M.extensions.content_blocks(
7028       options.contentBlocksLanguageMap)
7029     table.insert(extensions, content_blocks_extension)
7030   end
7031
7032   if options.definitionLists then
7033     local definition_lists_extension = M.extensions.definition_lists(
```

```
7034      options.tightLists)
7035    table.insert(extensions, definition_lists_extension)
7036  end
7037
7038  if options.fencedCode then
7039    local fenced_code_extension = M.extensions.fenced_code(
7040      options.blankBeforeCodeFence)
7041    table.insert(extensions, fenced_code_extension)
7042  end
7043
7044  if options.headerAttributes then
7045    local header_attributes_extension = M.extensions.header_attributes()
7046    table.insert(extensions, header_attributes_extension)
7047  end
7048
7049  if options.jekyllData then
7050    local jekyll_data_extension = M.extensions.jekyll_data(
7051      options.expectJekyllData)
7052    table.insert(extensions, jekyll_data_extension)
7053  end
7054
7055  if options.pipeTables then
7056    local pipe_tables_extension = M.extensions.pipe_tables(
7057      options.tableCaptions)
7058    table.insert(extensions, pipe_tables_extension)
7059  end
7060
7061  if options.strikeThrough then
7062    local strike_through_extension = M.extensions.strike_through()
7063    table.insert(extensions, strike_through_extension)
7064  end
7065
7066  if options.subscripts then
7067    local subscript_extension = M.extensions.subscripts()
7068    table.insert(extensions, subscript_extension)
7069  end
7070
7071  if options.superscripts then
7072    local superscript_extension = M.extensions.superscripts()
7073    table.insert(extensions, superscript_extension)
7074  end
7075
7076  if options.footnotes or options.inlineFootnotes then
7077    local footnotes_extension = M.extensions.footnotes(
7078      options.footnotes, options.inlineFootnotes)
7079    table.insert(extensions, footnotes_extension)
7080  end
```

```
7081
7082  if options.citations then
7083    local citations_extension = M.extensions.citations(options.citationNbsps)
7084    table.insert(extensions, citations_extension)
7085  end
7086
7087  if options.fancyLists then
7088    local fancy_lists_extension = M.extensions.fancy_lists()
7089    table.insert(extensions, fancy_lists_extension)
7090  end
```

Apply user-defined syntax extensions based on `options.extensions`.

```
7091  for _, user_extension_filename in ipairs(options.extensions) do
7092    local user_extension = (function(filename)
```

First, load and compile the contents of the user-defined syntax extension.

```
7093      local pathname = util.lookup_files(filename)
7094      local input_file = assert(io.open(pathname, "r"),
7095        [[Could not open user-defined syntax extension "]]
7096        .. pathname .. [[" for reading]])
7097      local input = assert(input_file:read("*a"))
7098      assert(input_file:close())
7099      local user_extension, err = load([[
7100        local sandbox = {}
7101        setmetatable(sandbox, {__index = _G})
7102        _ENV = sandbox
7103      ]] .. input)()
7104      assert(user_extension,
7105        [[Failed to compile user-defined syntax extension "]]
7106        .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
7107      assert(user_extension.api_version ~= nil,
7108        [[User-defined syntax extension "]] .. pathname
7109        .. [[" does not specify mandatory field "api_version"]])
7110      assert(type(user_extension.api_version) == "number",
7111        [[User-defined syntax extension "]] .. pathname
7112        .. [[" specifies field "api_version" of type "]]
7113        .. type(user_extension.api_version)
7114        .. [[" but "number" was expected]])
7115      assert(user_extension.api_version == metadata.user_extension_api_version,
7116        [[User-defined syntax extension "]] .. pathname
7117        .. [[" uses syntax extension API version "]]
7118        .. user_extension.api_version .. [[ but markdown.lua ]]
7119        .. metadata.version .. [[ uses API version ]]
7120        .. metadata.user_extension_api_version
7121        .. [[, which is incompatible]])
7122
```

```
      assert(user_extension.grammar_version ~= nil,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory field "grammar_version"]])
      assert(type(user_extension.grammar_version) == "number",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "grammar_version" of type "]]
        .. type(user_extension.grammar_version)
        .. [[" but "number" was expected]])
      assert(user_extension.grammar_version == metadata.grammar_version,
        [[User-defined syntax extension "]] .. pathname
        .. [[" uses grammar version "]] .. user_extension.grammar_version
        .. [[ but markdown.lua ]] .. metadata.version
        .. [[ uses grammar version ]] .. metadata.grammar_version
        .. [[, which is incompatible]])

      assert(user_extension.finalize_grammar ~= nil,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory "finalize_grammar" field]])
      assert(type(user_extension.finalize_grammar) == "function",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "finalize_grammar" of type "]]
        .. type(user_extension.finalize_grammar)
        .. [[" but "function" was expected]])
```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section **??**.)

```
      local extension = {
        extend_reader = user_extension.finalize_grammar,
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

Produce and return a conversion function from markdown to plain TEX.

```
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  local convert = reader.finalize_grammar(extensions)

  return convert
end

return M
```

3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.6.

```
7162
7163 local input
7164 if input_filename then
7165   local input_file = assert(io.open(input_filename, "r"),
7166     [[Could not open file "]] .. input_filename .. [[" for reading]])
7167   input = assert(input_file:read("*a"))
7168   assert(input_file:close())
7169 else
7170   input = assert(io.read("*a"))
7171 end
7172
```

First, ensure that the `options.cacheDir` directory exists.

```
7173 local lfs = require("lfs")
7174 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7175   assert(lfs.mkdir(options["cacheDir"]))
7176 end
7177
7178 local ran_ok, kpse = pcall(require, "kpse")
7179 if ran_ok then kpse.set_program_name("luatex") end
7180 local md = require("markdown")
```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```
7181 if metadata.version ~= md.metadata.version then
7182   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7183        "markdown.lua " .. md.metadata.version .. ".")
7184 end
7185 local convert = md.new(options)
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
7186 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7187
7188 if output_filename then
7189   local output_file = assert(io.open(output_filename, "w"),
7190     [[Could not open file "]] .. output_filename .. [[" for writing]])
7191   assert(output_file:write(output))
7192   assert(output_file:close())
7193 else
7194   assert(io.write(output))
7195 end
```

3.2 Plain TEX Implementation

The plain TEX implementation provides macros for the interfacing between TEX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain TEX exposed by the plain TEX interface (see Section 2.2).

3.2.1 Logging Facilities

```
\ifx\markdownInfo\undefined
  \def\markdownInfo#1{%
    \immediate\write-1{(l.\the\inputlineno) markdown.tex info: #1.}}%
\fi
\ifx\markdownWarning\undefined
  \def\markdownWarning#1{%
    \immediate\write16{(l.\the\inputlineno) markdown.tex warning: #1}}%
\fi
\ifx\markdownError\undefined
  \def\markdownError#1#2{%
    \errhelp{#2.}%
    \errmessage{(l.\the\inputlineno) markdown.tex error: #1}}%
\fi
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererLineBreakPrototype{\hfil\break}%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`\{}%
\def\markdownRendererRightBracePrototype{\char`\}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\&}%
\def\markdownRendererUnderscorePrototype{\char`_}%
\def\markdownRendererHashPrototype{\char`\#}%
\def\markdownRendererCircumflexPrototype{\char`^}%
\def\markdownRendererBackslashPrototype{\char`\\}%
\def\markdownRendererTildePrototype{\char`~}%
\def\markdownRendererPipePrototype{|}%
\def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
\def\markdownRendererLinkPrototype#1#2#3#4{#2}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \markdownInput{#3}}%
\def\markdownRendererContentBlockOnlineImagePrototype{%
  \markdownRendererImage}%
```

```
\def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
  \markdownRendererInputFencedCode{#3}{#2}}%
\def\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
\def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
\def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
\def\markdownRendererFancyOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
\def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{#1}%
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{\par}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{{\it#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input#1\relax{}}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \markdownRendererInputVerbatimPrototype{#1}}%
\def\markdownRendererHeadingOnePrototype#1{#1}%
\def\markdownRendererHeadingTwoPrototype#1{#1}%
\def\markdownRendererHeadingThreePrototype#1{#1}%
\def\markdownRendererHeadingFourPrototype#1{#1}%
\def\markdownRendererHeadingFivePrototype#1{#1}%
\def\markdownRendererHeadingSixPrototype#1{#1}%
\def\markdownRendererHorizontalRulePrototype{}%
\def\markdownRendererFootnotePrototype#1{#1}%
```

```
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
\def\markdownRendererTickedBoxPrototype{[X]}%
\def\markdownRendererHalfTickedBoxPrototype{[/]}%
\def\markdownRendererUntickedBoxPrototype{[ ]}%
\def\markdownRendererStrikeThroughPrototype#1{#1}%
\def\markdownRendererSuperscriptPrototype#1{#1}%
\def\markdownRendererSubscriptPrototype#1{#1}%
```

3.2.2.1 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p:

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth p.

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth p.

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth p.

```
\ExplSyntaxOn
\seq_new:N   \g_@@_jekyll_data_datatypes_seq
\tl_const:Nn \c_@@_jekyll_data_sequence_tl   { sequence }
\tl_const:Nn \c_@@_jekyll_data_mapping_tl    { mapping  }
\tl_const:Nn \c_@@_jekyll_data_scalar_tl     { scalar   }
```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```
\seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
\cs_new:Nn \markdown_jekyll_data_push_address_segment:n
  {
    \seq_if_empty:NF
      \g_@@_jekyll_data_datatypes_seq
      {
        \seq_get_right:NN
          \g_@@_jekyll_data_datatypes_seq
          \l_tmpa_tl
```

If we are currently in a sequence, we will put an asterisk (`*`) instead of a key into `\g_@@_jekyll_data_wildcard_absolute_address_seq` to make it represent a *wildcard*. Keeping a wildcard instead of a precise address makes it easy for the users to react to *any* item of a sequence regardless of how many there are, which can often be useful.

```
      \str_if_eq:NNTF
        \l_tmpa_tl
        \c_@@_jekyll_data_sequence_tl
        {
          \seq_put_right:Nn
            \g_@@_jekyll_data_wildcard_absolute_address_seq
            { *  }
        }
        {
          \seq_put_right:Nn
            \g_@@_jekyll_data_wildcard_absolute_address_seq
            { #1 }
        }
    }
}
```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (`/`) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```
\tl_new:N  \g_@@_jekyll_data_wildcard_absolute_address_tl
\tl_new:N  \g_@@_jekyll_data_wildcard_relative_address_tl
\cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
  {
```

```
    \seq_pop_left:NN #1 \l_tmpa_tl
    \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
    \seq_put_left:NV #1 \l_tmpa_tl
  }
\cs_new:Nn \markdown_jekyll_data_update_address_tls:
  {
    \markdown_jekyll_data_concatenate_address:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_absolute_address_tl
    \seq_get_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_relative_address_tl
  }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```
\cs_new:Nn \markdown_jekyll_data_push:nN
  {
    \markdown_jekyll_data_push_address_segment:n
      { #1 }
    \seq_put_right:NV
     \g_@@_jekyll_data_datatypes_seq
     #2
    \markdown_jekyll_data_update_address_tls:
  }
\cs_new:Nn \markdown_jekyll_data_pop:
  {
    \seq_pop_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \l_tmpa_tl
    \seq_pop_right:NN
      \g_@@_jekyll_data_datatypes_seq
      \l_tmpa_tl
    \markdown_jekyll_data_update_address_tls:
  }
```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```
\cs_new:Nn \markdown_jekyll_data_set_keyval:nn
  {
    \keys_set_known:nn
      { markdown/jekyllData }
      { { #1 } = { #2 } }
  }
\cs_generate_variant:Nn
  \markdown_jekyll_data_set_keyval:nn
  { Vn }
```

```
\cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
  {
    \markdown_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_scalar_tl
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_absolute_address_tl
      { #2 }
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_relative_address_tl
      { #2 }
    \markdown_jekyll_data_pop:
  }
```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```
\def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_sequence_tl
}
\def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_mapping_tl
}
\def\markdownRendererJekyllDataSequenceEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataMappingEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataBooleanPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\def\markdownRendererJekyllDataEmptyPrototype#1{}
\def\markdownRendererJekyllDataNumberPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\def\markdownRendererJekyllDataStringPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
```

```
}
\ExplSyntaxOff
```

3.2.3 Lua Snippets

After the \markdownPrepareLuaOptions macro has been fully expanded, the \markdownLuaOptions macro will expands to a Lua table that contains the plain TEX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```
\ExplSyntaxOn
\tl_new:N \g_@@_formatted_lua_options_tl
\cs_new:Nn \@@_format_lua_options:
  {
    \tl_gclear:N
      \g_@@_formatted_lua_options_tl
    \seq_map_function:NN
      \g_@@_lua_options_seq
      \@@_format_lua_option:n
  }
\cs_new:Nn \@@_format_lua_option:n
  {
    \@@_typecheck_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \bool_case_true:nF
      {
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_boolean_tl ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_number_tl ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_counter_tl
        }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { #1~=~  \l_tmpa_tl   ,~ }
          }
```

```
      {
        \str_if_eq_p:VV
          \l_tmpa_tl
          \c_@@_option_type_clist_tl
      }
        {
          \@@_get_option_value:nN
            { #1 }
            \l_tmpa_tl
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { #1~=~\c_left_brace_str }
          \clist_map_inline:Vn
            \l_tmpa_tl
            {
              \tl_gput_right:Nx
                \g_@@_formatted_lua_options_tl
                { "##1" ,~ }
            }
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { \c_right_brace_str ,~ }
        }
    }
    {
      \@@_get_option_value:nN
        { #1 }
        \l_tmpa_tl
      \tl_gput_right:Nx
        \g_@@_formatted_lua_options_tl
        { #1~=~ " \l_tmpa_tl " ,~ }
    }
  }
\cs_generate_variant:Nn
  \clist_map_inline:nn
  { Vn }
\let\markdownPrepareLuaOptions=\@@_format_lua_options:
\def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
\ExplSyntaxOff
```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain TEX. It exposes the `convert` function for the use by any further Lua code.

```
\def\markdownPrepare{%
```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```
  local lfs = require("lfs")
  local cacheDir = "\markdownOptionCacheDir"
```

```
  if not lfs.isdir(cacheDir) then
    assert(lfs.mkdir(cacheDir))
  end
```

Next, load the `markdown` module and create a converter function using the plain TEX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
  local md = require("markdown")
  local convert = md.new(\markdownLuaOptions)
}%
```

3.2.4 Buffering Markdown Input

The `\markdownIfOption{`⟨*name*⟩`}{`⟨*iftrue*⟩`}{`⟨*iffalse*⟩`}` macro is provided for testing, whether the value of `\markdownOption`⟨*name*⟩ is `true`. If the value is `true`, then ⟨*iftrue*⟩ is expanded, otherwise ⟨*iffalse*⟩ is expanded.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_if_option:nTF
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \msg_error:nnxx
          { @@ }
          { expected-boolean-option }
          { #1 }
          { \l_tmpa_tl }
      }
    \@@_get_option_value:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNTF
      \l_tmpa_tl
      \c_@@_option_value_true_tl
      { #2 }
      { #3 }
  }
\msg_new:nnn
  { @@ }
  { expected-boolean-option }
  {
    Option~#1~has~type~#2,~
    but~a~boolean~was~expected.
```

```
  }
\let\markdownIfOption=\@@_if_option:nTF
\ExplSyntaxOff
```

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
\csname newread\endcsname\markdownInputFileStream
\csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
\begingroup
  \catcode`\^^I=12%
  \gdef\markdownReadAndConvertTab{^^I}%
\endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the LaTeX 2$_\varepsilon$ `\filecontents` macro to plain TeX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `\markdownOptionStripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```
    |markdownIfOption{frozenCache}{}{@
      |immediate|openout|markdownOutputFileStream@
        |markdownOptionInputTempFileName|relax@
      |markdownInfo{Buffering markdown input into the temporary @
        input file "|markdownOptionInputTempFileName" and scanning @
        for the closing token sequence "#1"}@
    }@
```

Locally change the category of the special plain TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
    |def|do##1{|catcode`##1=12}|dospecials@
```

```
    |catcode`| =12@
    |markdownMakeOther@
```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stipping away leading percent signs (`%`) when `\markdownOptionStripPercentSigns` is enabled. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```
    |def|markdownReadAndConvertStripPercentSign##1{@
      |markdownIfOption{stripPercentSigns}{@
        |if##1%@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
        |else@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
            |expandafter|expandafter|expandafter##1@
        |fi@
      }{@
        |expandafter@
          |markdownReadAndConvertProcessLine@
          |expandafter##1@
      }@
    }@
```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```
    |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@
```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

When the ending token sequence appears in the line, make the next newline character close the `\markdownOptionInputTempFileName` file, return the character categories back to the former state, convert the `\markdownOptionInputTempFileName` file from markdown to plain TEX, `\input` the result of the conversion, and expand the ending control sequence.

```
        |def^^M{@
```

```
          |markdownInfo{The ending token sequence was found}@
          |markdownIfOption{frozenCache}{}{@
            |immediate|closeout|markdownOutputFileStream@
          }@
          |endgroup@
          |markdownInput{@
            |markdownOptionOutputDir@
            /|markdownOptionInputTempFileName@
          }@
          #2}@
      |fi@
```

Repeat with the next line.

```
      ^^M}@
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
    |catcode`|^^I=13@
    |def^^I{|markdownReadAndConvertTab}@
```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```
    |catcode`|^^M=13@
    |def^^M##1^^M{@
      |def^^M####1^^M{@
        |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
      ^^M}@
    ^^M}@
```

Reset the character categories back to the former state.

```
|endgroup
```

The following two sections of the implementation have been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
\ExplSyntaxOn
\int_compare:nT
  { \markdownMode = 3 }
  {
    \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
    \file_input:n { lt3luabridge.tex }
    \cs_new:Npn
      \markdownLuaExecute
      { \luabridgeExecute }
  }
\ExplSyntaxOff
```

3.2.5 Lua Shell Escape Bridge

The following TEX code is intended for TEX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the LuaTEX engine, their TEX distribution contains it, and uses shell access to produce and execute Lua scripts using the TEXLua interpreter [1, Section 4.1.1].

```
\ifnum\markdownMode<2\relax
\ifnum\markdownMode=0\relax
  \markdownWarning{Using mode 0: Shell escape via write18
                   (deprecated, to be removed in Markdown 3.0.0)}%
\else
  \markdownWarning{Using mode 1: Shell escape via os.execute
                   (deprecated, to be removed in Markdown 3.0.0)}%
\fi
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (`1`), disabled (`0`), or restricted (`2`).

Inherit the value of the the `\pdfshellescape` (LuaTEX, PdfTEX) or the `\shellescape` (XeTEX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```
\ifx\pdfshellescape\undefined
  \ifx\shellescape\undefined
    \ifnum\markdownMode=0\relax
      \def\markdownExecuteShellEscape{1}%
    \else
      \def\markdownExecuteShellEscape{%
        \directlua{tex.sprint(status.shell_escape or "1")}}%
    \fi
  \else
    \let\markdownExecuteShellEscape\shellescape
  \fi
\else
  \let\markdownExecuteShellEscape\pdfshellescape
\fi
```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```
\ifnum\markdownMode=0\relax
  \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
\else
```

```
  \def\markdownExecuteDirect#1{%
    \directlua{os.execute("\luaescapestring{#1}")}}%
\fi
```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```
\def\markdownExecute#1{%
  \ifnum\markdownExecuteShellEscape=1\relax
    \markdownExecuteDirect{#1}%
  \else
    \markdownError{I can not access the shell}{Either run the TeX
      compiler with the --shell-escape or the --enable-write18 flag,
      or set shell_escape=t in the texmf.cnf file}%
  \fi}%
```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TEX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```
  \catcode`|=0%
  \catcode`\\=12%
  |gdef|markdownLuaExecute#1{%
```

Create the file `\markdownOptionHelperScriptFileName` and fill it with the input Lua code prepended with kpathsea initialization, so that Lua modules from the TEX distribution are available.

```
    |immediate|openout|markdownOutputFileStream=%
      |markdownOptionHelperScriptFileName
    |markdownInfo{Writing a helper Lua script to the file
      "|markdownOptionHelperScriptFileName"}%
    |immediate|write|markdownOutputFileStream{%
      local ran_ok, error = pcall(function()
        local ran_ok, kpse = pcall(require, "kpse")
        if ran_ok then kpse.set_program_name("luatex") end
        #1
      end)
```

If there was an error, use the file `\markdownOptionErrorTempFileName` to store the error message.

```
      if not ran_ok then
        local file = io.open("%
          |markdownOptionOutputDir
          /|markdownOptionErrorTempFileName", "w")
        if file then
```

```
          file:write(error .. "\n")
          file:close()
        end
        print('\\markdownError{An error was encountered while executing
              Lua code}{For further clues, examine the file
              "|markdownOptionOutputDir
              /|markdownOptionErrorTempFileName"}')
      end}%
    |immediate|closeout|markdownOutputFileStream
```

Execute the generated `\markdownOptionHelperScriptFileName` Lua script using the TEXLua binary and store the output in the `\markdownOptionOutputTempFileName` file.

```
    |markdownInfo{Executing a helper Lua script from the file
      "|markdownOptionHelperScriptFileName" and storing the result in the
      file "|markdownOptionOutputTempFileName"}%
    |markdownExecute{texlua "|markdownOptionOutputDir
      /|markdownOptionHelperScriptFileName" > %
      "|markdownOptionOutputDir
      /|markdownOptionOutputTempFileName"}%
```

`\input` the generated `\markdownOptionOutputTempFileName` file.

```
    |input|markdownOptionOutputTempFileName|relax}%
|endgroup
```

3.2.6 Direct Lua Access

The following TEX code is intended for TEX engines that provide direct access to Lua (LuaTEX). The macro `\markdownLuaExecute` defined here and in Section 3.2.5 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```
\fi
\ifnum\markdownMode=2\relax
  \markdownWarning{Using mode 2: Direct Lua access
                   (deprecated, to be removed in Markdown 3.0.0)}%
```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.5,

```
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code.

```
  \catcode`|=0%
  \catcode`\\=12%
  |gdef|markdownLuaExecute#1{%
```

```
    |directlua{%
      local function print(input)
        local output = {}
        for line in input:gmatch("[^\r\n]+") do
          table.insert(output, line)
        end
        tex.print(output)
      end
      #1
    }%
  }%
|endgroup
\fi
```

3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TEX.

```
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code. Furthermore, use the ampersand symbol to specify parameters.

```
  \catcode`|=0%
  \catcode`\\=12%
  \catcode`|&=6%
  |gdef|markdownInput#1{%
```

Change the category code of the percent sign (`%`) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |begingroup
    |catcode`|%=12
```

Furthermore, also change the category code of the hash sign (`#`) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
    |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache`⟨*number*⟩ macro, and increment `frozenCacheCounter`.

```
    |markdownIfOption{frozenCache}{%
      |ifnum|markdownOptionFrozenCacheCounter=0|relax
        |markdownInfo{Reading frozen cache from
          "|markdownOptionFrozenCacheFileName"}%
        |input|markdownOptionFrozenCacheFileName|relax
```

```
      |fi
      |markdownInfo{Including markdown document number
        "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
      |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }{%
      |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as LaTeXMk to track changes to the markdown document.

```
      |openin|markdownInputFileStream&1
      |closein|markdownInputFileStream
      |markdownPrepareLuaOptions
      |markdownLuaExecute{%
        |markdownPrepare
        local file = assert(io.open("&1", "r"),
          [[Could not open file "&1" for reading]])
        local input = assert(file:read("*a"))
        assert(file:close())
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
        print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%
```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }%
    |endgroup
  }%
|endgroup
```

3.3 LaTeX Implementation

The LaTeX implemenation makes use of the fact that, apart from some subtle differences, LaTeX implements the majority of the plain TeX format [10, Section 9]. As a consequence, we can directly reuse the existing plain TeX implementation.

```
\def\markdownVersionSpace{ }%
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \markdownVersion\markdownVersionSpace markdown renderer]%
```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```
\ExplSyntaxOn
\@@_latex_define_renderers:
\@@_latex_define_renderer_prototypes:
\ExplSyntaxOff
```

3.3.1 Logging Facilities

The LaTeX implementation redefines the plain TeX logging macros (see Section 3.2.1) to use the LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the LaTeX interface (see Section 2.3.2).

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

The `markdown`, and `markdown*` LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```
\renewenvironment{markdown}{%
  \markdownReadAndConvert@markdown{}}{%
  \markdownEnd}%
\renewenvironment{markdown*}[1]{%
  \markdownSetup{#1}%
  \markdownReadAndConvert@markdown*}{%
  \markdownEnd}%
\begingroup
```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```
  \catcode`\|=0\catcode`\<=1\catcode`\>=2%
  \catcode`\\=12|catcode`|{=12|catcode`|}=12%
  |gdef|markdownReadAndConvert@markdown#1<%
    |markdownReadAndConvert<\end{markdown#1}>%
                           <|end<markdown#1>>>%
|endgroup
```

3.3.2.1 LaTeX Themes

This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
\ExplSyntaxOn
```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```
\newcommand\markdownLaTeXThemeName{}
\seq_new:N \g_@@_latex_themes_seq
\seq_put_right:NV
  \g_@@_latex_themes_seq
  \markdownLaTeXThemeName
\newcommand\markdownLaTeXThemeLoad[2]{
  \def\@tempa{%
    \def\markdownLaTeXThemeName{#2}
    \seq_put_right:NV
      \g_@@_latex_themes_seq
      \markdownLaTeXThemeName
    \RequirePackage{#1}
    \seq_pop_right:NN
      \g_@@_latex_themes_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_latex_themes_seq
      \l_tmpa_tl
    \exp_args:NNV
      \def
      \markdownLaTeXThemeName
      \l_tmpa_tl}
  \ifmarkdownLaTeXLoaded
    \@tempa
  \else
    \exp_args:No
      \AtEndOfPackage
      { \@tempa }
  \fi}
\ExplSyntaxOff
```

The witiko/dot theme enables the fencedCode Lua option:

```
\markdownSetup{fencedCode}%
```

We load the ifthen and grffile packages, see also Section 1.1.3:

```
\RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
\let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
  \markdownRendererInputFencedCodePrototype
```

If the infostring starts with dot …, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the \markdownOptionFrozenCache plain TEX option is disabled and the code block has not been previously typeset:

```
\renewcommand\markdownRendererInputFencedCode[2]{%
  \def\next##1 ##2\relax{%
    \ifthenelse{\equal{##1}{dot}}{%
```

```
      \markdownIfOption{frozenCache}{}{%
        \immediate\write18{%
          if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
          then
            dot -Tpdf -o #1.pdf #1;
            cp #1 #1.pdf.source;
          fi}}%
```

We include the typeset image using the image token renderer:

```
      \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%
```

If the infostring does not start with `dot …`, we use the previous definition of the fenced code token renderer prototype:

```
    }{%
      \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
    }%
  }%
  \next#2 \relax}%
```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```
\let\markdown@witiko@graphicx@http@oldRendererImagePrototype
  \markdownRendererImagePrototype
```

We load the catchfile and grffile packages, see also Section 1.1.3:

```
\RequirePackage{catchfile,grffile}
```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```
\newcount\markdown@witiko@graphicx@http@counter
\markdown@witiko@graphicx@http@counter=0
\newcommand\markdown@witiko@graphicx@http@filename{%
  \markdownOptionCacheDir/witiko_graphicx_http%
  .\the\markdown@witiko@graphicx@http@counter}%
```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to downloads the online image to the pathname.

```
\newcommand\markdown@witiko@graphicx@http@download[2]{%
  wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}
```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
\begingroup
\catcode`\%=12
\catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
\global\def\markdownRendererImagePrototype#1#2#3#4{^^A
  \begingroup
    \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `\markdownOptionFrozenCache` plain TEX option is disabled:

```
    \markdownIfOption{frozenCache}{}{^^A
      \immediate\write18{^^A
        mkdir -p "\markdownOptionCacheDir";
        if printf '%s' "#3" | grep -q -E '^https?:';
        then
```

The image will be downloaded to the pathname `\markdownOptionCacheDir`/⟨*the MD5 digest of the image URL*⟩.⟨*the suffix of the image URL*⟩:

```
          OUTPUT_PREFIX="\markdownOptionCacheDir";
          OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
          OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]//')";
          OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
          if ! [ -e "$OUTPUT" ];
          then
            \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
            printf '%s' "$OUTPUT" > "\filename";
          fi;
```

If the image does not have the http or https protocols or the image has already been downloaded, the URL will be stored as-is:

```
        else
          printf '%s' '#3' > "\filename";
        fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
    \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
    \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
      {#1}{#2}{\filename}{#4}^^A
  \endgroup
  \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
\endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
\renewcommand\markdownRendererTildePrototype{~}%
```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
\DeclareOption*{%
  \expandafter\markdownSetup\expandafter{\CurrentOption}}%
\ProcessOptions\relax
```

After processing the options, activate the `jekyllDataRenderes`, `renderers`, `rendererPrototypes`, and `code` keys.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/latex-options }
  {
    renderers .code:n = {
      \keys_set:nn
        { markdown/latex-options/renderers }
        { #1 }
    },
    rendererPrototypes .code:n = {
      \keys_set:nn
        { markdown/latex-options/renderer-prototypes }
        { #1 }
    },
```

The `code` key is used to immediately expand and execute code, which can be especially useful in LaTeX setup snippets.

```
    code .code:n = { #1 },
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key–values (see Section 2.2.4.1) without using the expl3 language.

```
    jekyllDataRenderers .code:n = {
      \keys_set:nn
        { markdown/latex-options/jekyll-data-renderers }
        { #1 }
    },
  }
\keys_define:nn
  { markdown/latex-options/jekyll-data-renderers }
  {
    unknown .code:n = {
      \tl_set_eq:NN
        \l_tmpa_tl
        \l_keys_key_str
      \tl_put_right:Nn
        \l_tmpa_tl
        {
          .code:n = { #1 }
```

```
        }
      \keys_define:nV
        { markdown/jekyllData }
        \l_tmpa_tl
    }
  }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\ExplSyntaxOff
```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.1), none of it will take effect.

```
\markdownIfOption{plain}{\iffalse}{\iftrue}
```

If the `tightLists` Lua option is disabled or the current document class is beamer, do not load the paralist package.

```
\markdownIfOption{tightLists}{
  \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}%
}{}
```

If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
\ExplSyntaxOn
\@ifpackageloaded{paralist}{
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_number_style_tl
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_delimiter_style_tl
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_number:nn
    {
      \str_case:nn
        { #1 }
        {
          { Decimal } { #2 }
          { LowerRoman } { \int_to_roman:n { #2 } }
          { UpperRoman } { \int_to_Roman:n { #2 } }
          { LowerAlpha } { \int_to_alph:n { #2 } }
          { UpperAlpha } { \int_to_alph:n { #2 } }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label_delimiter:n
```

```
    {
      \str_case:nn
        { #1 }
        {
          { Default } { . }
          { OneParen } { ) }
          { Period } { . }
        }
    }
  \cs_new:Nn
    \@@_latex_fancy_list_item_label:nnn
    {
      \@@_latex_fancy_list_item_label_number:nn
        { #1 }
        { #3 }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \cs_new:Nn
    \@@_latex_paralist_style:nn
    {
      \str_case:nn
        { #1 }
        {
          { Decimal } { 1 }
          { LowerRoman } { i }
          { UpperRoman } { I }
          { LowerAlpha } { a }
          { UpperAlpha } { A }
        }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\begin{compactitem}},
    ulEndTight = {\end{compactitem}},
    fancyOlBegin = {
      \group_begin:
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \tl_set:Nn
        \l_tmpa_tl
        { \begin{enumerate}[ }
```

```
      \tl_put_right:Nx
        \l_tmpa_tl
        { \@@_latex_paralist_style:nn { #1 } { #2 } }
      \tl_put_right:Nn
        \l_tmpa_tl
        { ] }
      \l_tmpa_tl
    },
    fancyOlEnd = {
      \end{enumerate}
      \group_end:
    },
    olBeginTight = {\begin{compactenum}},
    olEndTight = {\end{compactenum}},
    fancyOlBeginTight = {
      \group_begin:
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \tl_set:Nn
        \l_tmpa_tl
        { \begin{compactenum}[ }
      \tl_put_right:Nx
        \l_tmpa_tl
        { \@@_latex_paralist_style:nn { #1 } { #2 } }
      \tl_put_right:Nn
        \l_tmpa_tl
        { ] }
      \l_tmpa_tl
    },
    fancyOlEndTight = {
      \end{compactenum}
      \group_end:
    },
    fancyOlItemWithNumber = {
      \item
        [
          \@@_latex_fancy_list_item_label:VVn
            \l_@@_latex_fancy_list_item_label_number_style_tl
            \l_@@_latex_fancy_list_item_label_delimiter_style_tl
            { #1 }
        ]
    },
    dlBeginTight = {\begin{compactdesc}},
```

```
    dlEndTight = {\end{compactdesc}}}}
  \cs_generate_variant:Nn
    \@@_latex_fancy_list_item_label:nnn
    { VVn }
}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\markdownRendererUlBegin},
    ulEndTight = {\markdownRendererUlEnd},
    fancyOlBegin = {\markdownRendererOlBegin},
    fancyOlEnd = {\markdownRendererOlEnd},
    olBeginTight = {\markdownRendererOlBegin},
    olEndTight = {\markdownRendererOlEnd},
    fancyOlBeginTight = {\markdownRendererOlBegin},
    fancyOlEndTight = {\markdownRendererOlEnd},
    dlBeginTight = {\markdownRendererDlBegin},
    dlEndTight = {\markdownRendererDlEnd}}}
}
\ExplSyntaxOff
\RequirePackage{amsmath,ifthen}
```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```
\@ifpackageloaded{unicode-math}{
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\mdlgwhtsquare$},
  }}
}{
  \RequirePackage{amssymb}
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\square$},
  }}
}
\RequirePackage{csvsimple}
\RequirePackage{fancyvrb}
\RequirePackage{graphicx}
\markdownSetup{rendererPrototypes={
  lineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
```

We can capitalize on the fact that the expansion of renderers is performed by TEX during the typesetting. Therefore, even if we don't know whether a span of text is

part of math formula or not when we are parsing markdown,[8] we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```
  codeSpan = {%
    \ifmmode
      \text{#1}%
    \else
      \texttt{#1}%
    \fi
  },
  contentBlock = {%
    \ifthenelse{\equal{#1}{csv}}{%
      \begin{table}%
        \begin{center}%
          \csvautotabular{#3}%
        \end{center}
        \ifx\empty#4\empty\else
          \caption{#4}%
        \fi
      \end{table}%
    }{%
      \ifthenelse{\equal{#1}{tex}}{%
        \catcode`\%=14\relax
        \catcode`\#=6\relax
        \input #3\relax
        \catcode`\%=12\relax
        \catcode`\#=12\relax
      }{%
        \markdownInput{#3}%
      }%
    }%
  },
  image = {%
    \begin{figure}%
      \begin{center}%
        \includegraphics{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
    \end{figure}},
  ulBegin = {\begin{itemize}},
```

[8] This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```
  ulEnd = {\end{itemize}},
  olBegin = {\begin{enumerate}},
  olItem = {\item{}},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  dlBegin = {\begin{description}},
  dlItem = {\item[#1]},
  dlEnd = {\end{description}},
  emphasis = {\emph{#1}},
  tickedBox = {$\boxtimes$},
  halfTickedBox = {$\boxdot$},
```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```
  headerAttributeContextBegin = {
    \markdownSetup{
      rendererPrototypes = {
        attributeIdentifier = {%
          \begingroup
          \def\next####1{%
            \def####1########1{%
              \endgroup
              ####1{########1}%
              \label{##1}%
            }%
          }%
          \next\markdownRendererHeadingOne
          \next\markdownRendererHeadingTwo
          \next\markdownRendererHeadingThree
          \next\markdownRendererHeadingFour
          \next\markdownRendererHeadingFive
          \next\markdownRendererHeadingSix
        },
      },
    }%
  },
  superscript = {\textsuperscript{#1}},
  subscript = {\textsubscript{#1}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  inputFencedCode = {%
    \ifx\relax#2\relax
      \VerbatimInput{#1}%
    \else
      \@ifundefined{minted@code}{%
        \@ifundefined{lst@version}{%
```

```
          \markdownRendererInputFencedCode{#1}{}%
```

When the listings package is loaded, use it for syntax highlighting.

```
        }{%
          \lstinputlisting[language=#2]{#1}%
        }%
```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```
      }{%
        \catcode`\#=6\relax
        \inputminted{#2}{#1}%
        \catcode`\#=12\relax
      }%
    \fi},
  horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  footnote = {\footnote{#1}}}}
```

Support the nesting of strong emphasis.

```
\ExplSyntaxOn
\def\markdownLATEXStrongEmphasis#1{%
  \str_if_in:NnTF
    \f@series
    { b }
    { \textnormal{#1} }
    { \textbf{#1} }
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={strongEmphasis={%
  \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
\@ifundefined{chapter}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}\leavevmode},
    headingFive = {\subparagraph{#1}\leavevmode}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}\leavevmode},
    headingSix = {\subparagraph{#1}\leavevmode}}}
}%
```

3.3.4.1 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```
\markdownSetup{
  rendererPrototypes = {
    ulItem = {%
      \futurelet\markdownLaTeXCheckbox\markdownLaTeXUlItem
    },
  },
}
\def\markdownLaTeXUlItem{%
  \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
    \item[\markdownLaTeXCheckbox]%
    \expandafter\@gobble
  \else
    \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
      \item[\markdownLaTeXCheckbox]%
      \expandafter\expandafter\expandafter\@gobble
    \else
      \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
        \item[\markdownLaTeXCheckbox]%
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\@gobble
      \else
        \item{}%
      \fi
    \fi
  \fi
}
```

3.3.4.2 HTML elements

If the `html` option is enabled and we are using TEX4ht[9], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
          \IgnorePar
```

[9] See https://tug.org/tex4ht/.

```
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

3.3.4.3 Citations

Here is a basic implementation for citations that uses the LaTeX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibLaTeX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\RequirePackage{gobble}
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \cite{#1#2#6}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobblethree
      \fi
    \else%  Before a postnote (#5), dump the accumulator
      \ifx\relax#1\relax\else
        \cite{#1}%
      \fi
      \cite[#5]{#6}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
```

```
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobblethree
    \fi
  \else%  Before a prenote (#4), dump the accumulator
    \ifx\relax#1\relax\else
      \cite{#1}%
    \fi
    \ifnum\markdownLaTeXCitationsCounter>1\relax
      \space  % Insert a space before the prenote in later citations
    \fi
    #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXBasicCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobblethree
  \fi\markdownLaTeXBasicCitations{#1#2#6},}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% Natbib implementation
\def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citep{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  Before a postnote (#4), dump the accumulator
      \ifx\relax#1\relax\else
        \citep{#1}%
      \fi
      \citep[][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
```

```
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  Before a prenote (#3), dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citep{#1}%
    \fi
    \citep[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibCitations{#1,#5}}
\def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citet{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  After a prenote or a postnote, dump the accumulator
      \ifx\relax#1\relax\else
        \citet{#1}%
      \fi
      , \citet[#3][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
        ,
      \else
```

```
        \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
          ,
        \fi
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibTextCitations
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  After a prenote or a postnote, dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citet{#1}%
    \fi
    , \citet[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
      ,
    \else
      \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
        ,
      \fi
    \fi
    \expandafter\expandafter\expandafter
    \markdownLaTeXNatbibTextCitations
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibTextCitations{#1,#5}}

% BibLaTeX implementation
\def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \autocites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
\def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \textcites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
```

```
8402
8403 \markdownSetup{rendererPrototypes = {
8404   cite = {%
8405     \markdownLaTeXCitationsCounter=1%
8406     \def\markdownLaTeXCitationsTotal{#1}%
8407     \@ifundefined{autocites}{%
8408       \@ifundefined{citep}{%
8409         \expandafter\expandafter\expandafter
8410         \markdownLaTeXBasicCitations
8411         \expandafter\expandafter\expandafter{%
8412         \expandafter\expandafter\expandafter}%
8413         \expandafter\expandafter\expandafter{%
8414         \expandafter\expandafter\expandafter}%
8415       }{%
8416         \expandafter\expandafter\expandafter
8417         \markdownLaTeXNatbibCitations
8418         \expandafter\expandafter\expandafter{%
8419         \expandafter\expandafter\expandafter}%
8420       }%
8421     }{%
8422       \expandafter\expandafter\expandafter
8423       \markdownLaTeXBibLaTeXCitations
8424       \expandafter{\expandafter}%
8425     }},
8426   textCite = {%
8427     \markdownLaTeXCitationsCounter=1%
8428     \def\markdownLaTeXCitationsTotal{#1}%
8429     \@ifundefined{autocites}{%
8430       \@ifundefined{citep}{%
8431         \expandafter\expandafter\expandafter
8432         \markdownLaTeXBasicTextCitations
8433         \expandafter\expandafter\expandafter{%
8434         \expandafter\expandafter\expandafter}%
8435         \expandafter\expandafter\expandafter{%
8436         \expandafter\expandafter\expandafter}%
8437       }{%
8438         \expandafter\expandafter\expandafter
8439         \markdownLaTeXNatbibTextCitations
8440         \expandafter\expandafter\expandafter{%
8441         \expandafter\expandafter\expandafter}%
8442       }%
8443     }{%
8444       \expandafter\expandafter\expandafter
8445       \markdownLaTeXBibLaTeXTextCitations
8446       \expandafter{\expandafter}%
8447     }}}}
```

3.3.4.4 Links

Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```
\RequirePackage{url}
\RequirePackage{expl3}
\ExplSyntaxOn
\def\markdownRendererLinkPrototype#1#2#3#4{
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_set:Nn \l_tmpb_tl { #2 }
  \bool_set:Nn
    \l_tmpa_bool
    {
      \tl_if_eq_p:NN
        \l_tmpa_tl
        \l_tmpb_tl
    }
  \tl_set:Nn \l_tmpa_tl { #4 }
  \bool_set:Nn
    \l_tmpb_bool
    {
      \tl_if_empty_p:N
        \l_tmpa_tl
    }
```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```
  \bool_if:nTF
    {
      \l_tmpa_bool && \l_tmpb_bool
    }
    {
      \markdownLaTeXRendererAutolink { #2 } { #3 }
    }{
      \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
    }
}
\def\markdownLaTeXRendererAutolink#1#2{%
```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```
  \tl_set:Nn
    \l_tmpa_tl
    { #2 }
  \tl_trim_spaces:N
    \l_tmpa_tl
  \tl_set:Nx
```

```
    \l_tmpb_tl
    {
      \tl_range:Nnn
        \l_tmpa_tl
        { 1 }
        { 1 }
    }
  \str_if_eq:NNTF
    \l_tmpb_tl
    \c_hash_str
    {
      \tl_set:Nx
        \l_tmpb_tl
        {
          \tl_range:Nnn
            \l_tmpa_tl
            { 2 }
            { -1 }
        }
      \exp_args:NV
        \ref
        \l_tmpb_tl
    }{
      \url { #2 }
    }
}
\ExplSyntaxOff
\def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
  #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

3.3.4.5 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```
\newcount\markdownLaTeXRowCounter
\newcount\markdownLaTeXRowTotal
\newcount\markdownLaTeXColumnCounter
\newcount\markdownLaTeXColumnTotal
\newtoks\markdownLaTeXTable
\newtoks\markdownLaTeXTableAlignment
\newtoks\markdownLaTeXTableEnd
\AtBeginDocument{%
  \@ifpackageloaded{booktabs}{%
    \def\markdownLaTeXTopRule{\toprule}%
    \def\markdownLaTeXMidRule{\midrule}%
    \def\markdownLaTeXBottomRule{\bottomrule}%
  }{%
    \def\markdownLaTeXTopRule{\hline}%
```

```
    \def\markdownLaTeXMidRule{\hline}%
    \def\markdownLaTeXBottomRule{\hline}%
  }%
}
\markdownSetup{rendererPrototypes={
  table = {%
    \markdownLaTeXTable={}%
    \markdownLaTeXTableAlignment={}%
    \markdownLaTeXTableEnd={%
      \markdownLaTeXBottomRule
      \end{tabular}}%
    \ifx\empty#1\empty\else
      \addto@hook\markdownLaTeXTable{%
        \begin{table}
        \centering}%
      \addto@hook\markdownLaTeXTableEnd{%
        \caption{#1}
        \end{table}}%
    \fi
    \addto@hook\markdownLaTeXTable{\begin{tabular}}%
    \markdownLaTeXRowCounter=0%
    \markdownLaTeXRowTotal=#2%
    \markdownLaTeXColumnTotal=#3%
    \markdownLaTeXRenderTableRow
  }
}}
\def\markdownLaTeXRenderTableRow#1{%
  \markdownLaTeXColumnCounter=0%
  \ifnum\markdownLaTeXRowCounter=0\relax
    \markdownLaTeXReadAlignments#1%
    \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
        \the\markdownLaTeXTableAlignment}}%
    \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
  \else
    \markdownLaTeXRenderTableCell#1%
  \fi
  \ifnum\markdownLaTeXRowCounter=1\relax
    \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
  \fi
  \advance\markdownLaTeXRowCounter by 1\relax
  \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
    \the\markdownLaTeXTable
    \the\markdownLaTeXTableEnd
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableRow}
\def\markdownLaTeXReadAlignments#1{%
```

```
  \advance\markdownLaTeXColumnCounter by 1\relax
  \if#1d%
    \addto@hook\markdownLaTeXTableAlignment{l}%
  \else
    \addto@hook\markdownLaTeXTableAlignment{#1}%
  \fi
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
    \expandafter\@gobble
  \fi\markdownLaTeXReadAlignments}
\def\markdownLaTeXRenderTableCell#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
    \addto@hook\markdownLaTeXTable{#1&}%
  \else
    \addto@hook\markdownLaTeXTable{#1\\}%
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableCell}
\fi
```

3.3.4.6 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  {
    author  .code:n = { \author{#1} },
    date    .code:n = { \date{#1}   },
    title   .code:n = { \title{#1}  },
  }
```

To complement the default setup of our key–values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```
% TODO: Remove the command definition in TeX Live 2021.
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\markdownSetup{
  rendererPrototypes = {
    jekyllDataEnd = {
%     TODO: Remove the else branch in TeX Live 2021.
      \IfFormatAtLeastTF
        { 2020-10-01 }
        { \AddToHook{begindocument/end}{\maketitle} }
        {
          \ifx\@onlypreamble\@notprerr
```

```
            % We are in the document
            \maketitle
          \else
            % We are in the preamble
            \RequirePackage{etoolbox}
            \AfterEndPreamble{\maketitle}
          \fi
        }
    },
  },
}
\ExplSyntaxOff
```

3.3.4.7 Strike-Through

If the `strikeThrough` option is enabled, we will load the soulutf8 package and use it to implement strike-throughs.

```
\markdownIfOption{strikeThrough}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
    rendererPrototypes = {
      strikeThrough = {%
        \st{#1}%
      },
    }
  }
}{}
```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```
\newcommand\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat}%
```

3.4 ConTEXt Implementation

The ConTEXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTEXt formats *seem* to implement (the documentation is scarce) the majority of the plain TEX format required by the plain

TEX implementation. As a consequence, we can directly reuse the existing plain TEX implementation after supplying the missing plain TEX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents LATEX package.

```
\def\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTEXt.

```
  \catcode`|=12}%
```

3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConTEXt interface (see Section 2.4.2).

```
\long\def\inputmarkdown{%
  \dosingleempty
  \doinputmarkdown}%
\long\def\doinputmarkdown[#1]#2{%
  \begingroup
    \iffirstargument
      \setupmarkdown{#1}%
    \fi
    \markdownInput{#2}%
  \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's TEX, trailing spaces are removed very early on when a line is being put to the input buffer. [11, sec. 31]. According to Eijkhout [12, sec. 2.2], this is because "these spaces are hard to see in an editor". At the moment, there is no option to suppress this behavior in (Lua)TEX, but ConTEXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTEXt MkIV and therefore to insert hard line breaks into markdown text.

```
\ifx\startluacode\undefined  % MkII
  \begingroup
    \catcode`\|=0%
    \catcode`\\=12%
    |gdef|startmarkdown{%
```

```
      |markdownReadAndConvert{\stopmarkdown}%
                               {|stopmarkdown}}%
    |gdef|stopmarkdown{%
      |markdownEnd}%
  |endgroup
\else  % MkIV
  \startluacode
    document.markdown_buffering = false
    local function preserve_trailing_spaces(line)
      if document.markdown_buffering then
        line = line:gsub("[ \t][ \t]$", "\t\t")
      end
      return line
    end
    resolvers.installinputlinehandler(preserve_trailing_spaces)
  \stopluacode
  \begingroup
    \catcode`\|=0%
    \catcode`\\=12%
    |gdef|startmarkdown{%
      |ctxlua{document.markdown_buffering = true}%
      |markdownReadAndConvert{\stopmarkdown}%
                               {|stopmarkdown}}%
    |gdef|stopmarkdown{%
      |ctxlua{document.markdown_buffering = false}%
      |markdownEnd}%
  |endgroup
\fi
```

3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\def\markdownRendererLineBreakPrototype{\blank}%
\def\markdownRendererLeftBracePrototype{\textbraceleft}%
\def\markdownRendererRightBracePrototype{\textbraceright}%
\def\markdownRendererDollarSignPrototype{\textdollar}%
\def\markdownRendererPercentSignPrototype{\percent}%
\def\markdownRendererUnderscorePrototype{\textunderscore}%
\def\markdownRendererCircumflexPrototype{\textcircumflex}%
\def\markdownRendererBackslashPrototype{\textbackslash}%
\def\markdownRendererTildePrototype{\textasciitilde}%
\def\markdownRendererPipePrototype{\char`|}%
\def\markdownRendererLinkPrototype#1#2#3#4{%
  \useURL[#1][#3][][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
  \fi\tt<\hyphenatedurl{#3}>}}%
\usemodule[database]
\defineseparatedlist
```

```
  [MarkdownConTeXtCSV]
  [separator={,},
   before=\bTABLE,after=\eTABLE,
   first=\bTR,last=\eTR,
   left=\bTD,right=\eTD]
\def\markdownConTeXtCSV{csv}
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \def\markdownConTeXtCSV@arg{#1}%
  \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
    \placetable[][tab:#1]{#4}{%
      \processseparatedfile[MarkdownConTeXtCSV][#3]}%
  \else
    \markdownInput{#3}%
  \fi}%
\def\markdownRendererImagePrototype#1#2#3#4{%
  \placefigure[][]{#4}{\externalfigure[#3]}}%
\def\markdownRendererUlBeginPrototype{\startitemize}%
\def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
\def\markdownRendererUlItemPrototype{\item}%
\def\markdownRendererUlEndPrototype{\stopitemize}%
\def\markdownRendererUlEndTightPrototype{\stopitemize}%
\def\markdownRendererOlBeginPrototype{\startitemize[n]}%
\def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
\def\markdownRendererOlItemPrototype{\item}%
\def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
\def\markdownRendererOlEndPrototype{\stopitemize}%
\def\markdownRendererOlEndTightPrototype{\stopitemize}%
\definedescription
  [MarkdownConTeXtDlItemPrototype]
  [location=hanging,
   margin=standard,
   headstyle=bold]%
\definestartstop
  [MarkdownConTeXtDlPrototype]
  [before=\blank,
   after=\blank]%
\definestartstop
  [MarkdownConTeXtDlTightPrototype]
  [before=\blank\startpacked,
   after=\stoppacked\blank]%
\def\markdownRendererDlBeginPrototype{%
  \startMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlBeginTightPrototype{%
  \startMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererDlItemPrototype#1{%
  \startMarkdownConTeXtDlItemPrototype{#1}}%
\def\markdownRendererDlItemEndPrototype{%
```

```
  \stopMarkdownConTeXtDlItemPrototype}%
\def\markdownRendererDlEndPrototype{%
  \stopMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlEndTightPrototype{%
  \stopMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererEmphasisPrototype#1{{\em#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
\def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
\def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \ifx\relax#2\relax
    \typefile{#1}%
  \else
```

The code fence infostring is used as a name from the ConTEXt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```
\definetyping [latex]
\setuptyping  [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext
```

```
    \typefile[#2][]{#1}%
  \fi}%
\def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
\def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
\def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
\def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
\def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
\def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
\def\markdownRendererHorizontalRulePrototype{%
  \blackrule[height=1pt, width=\hsize]}%
\def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
\def\markdownRendererTickedBoxPrototype{$\boxtimes$}
\def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
\def\markdownRendererUntickedBoxPrototype{$\square$}
```

```
\def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
\def\markdownRendererSuperscriptPrototype#1{\high{#1}}
\def\markdownRendererSubscriptPrototype#1{\low{#1}}
```

3.4.2.1 Tables

There is a basic implementation of tables.

```
\newcount\markdownConTeXtRowCounter
\newcount\markdownConTeXtRowTotal
\newcount\markdownConTeXtColumnCounter
\newcount\markdownConTeXtColumnTotal
\newtoks\markdownConTeXtTable
\newtoks\markdownConTeXtTableFloat
\def\markdownRendererTablePrototype#1#2#3{%
  \markdownConTeXtTable={}%
  \ifx\empty#1\empty
    \markdownConTeXtTableFloat={%
      \the\markdownConTeXtTable}%
  \else
    \markdownConTeXtTableFloat={%
      \placetable{#1}{\the\markdownConTeXtTable}}%
  \fi
  \begingroup
  \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
  \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
  \setupTABLE[r][1][topframe=on, bottomframe=on]
  \setupTABLE[r][#1][bottomframe=on]
  \markdownConTeXtRowCounter=0%
  \markdownConTeXtRowTotal=#2%
  \markdownConTeXtColumnTotal=#3%
  \markdownConTeXtRenderTableRow}
\def\markdownConTeXtRenderTableRow#1{%
  \markdownConTeXtColumnCounter=0%
  \ifnum\markdownConTeXtRowCounter=0\relax
    \markdownConTeXtReadAlignments#1%
    \markdownConTeXtTable={\bTABLE}%
  \else
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\bTR}%
    \markdownConTeXtRenderTableCell#1%
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTR}%
  \fi
  \advance\markdownConTeXtRowCounter by 1\relax
  \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTABLE}%
    \the\markdownConTeXtTableFloat
```

```
    \endgroup
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtRenderTableRow}
\def\markdownConTeXtReadAlignments#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \if#1d%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1l%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1c%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
  \fi\if#1r%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
  \fi
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtReadAlignments}
\def\markdownConTeXtRenderTableCell#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \markdownConTeXtTable=\expandafter{%
    \the\markdownConTeXtTable\bTD#1\eTD}%
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtRenderTableCell}
\stopmodule\protect
```

References

[1] LuaTEX development team. *LuaTEX reference manual*. Version 1.10 (stable). July 23, 2021. URL: https://www.pragma-ade.com/general/manuals/luatex.pdf (visited on 09/30/2022).

[2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[4] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[5] Frank Mittelbach. *The `doc` and `shortvrb` Packages*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/doc.pdf (visited on 02/19/2018).

[6] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf (visited on 02/11/2021).

[7] Vít Novotný. *LaTeX 2ε no longer keys packages by pathnames*. Feb. 20, 2021. URL: https://github.com/latex3/latex2e/issues/510 (visited on 02/21/2021).

[8] Geoffrey M. Poore. *The minted Package. Highlighted source code in LaTeX*. July 19, 2017. URL: https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf (visited on 09/01/2020).

[9] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[10] Johannes Braams et al. *The LaTeX 2ε Sources*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/source2e.pdf (visited on 01/08/2018).

[11] Donald Ervin Knuth. *TeX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.

[12] Victor Eijkhout. *TeX by Topic. A TeXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

Index